Modelling arithmetic strategies

Thesis

Modelling Arithmetic Strategies

Roshni Devi

Thesis submitted in partial fulfilment of requirements for the degree of Doctor of Philosophy in Cognitive Science

The Open University
Milton Keynes
U.K.

March, 1991.

Abstract

This thesis examines children's arithmetic strategies and their relation to the concepts of commutativity and associativity. Two complementary methods were used in this research: empirical studies and computational models.

Empirical studies were carried out to identify the strategies children used for solving problems like 3 + 4, and 3 + 4 + 7, and the conceptual knowledge associated with them. Their understanding of subtraction problems where the minuend is less than the subtrahend (e.g. 6 - 8) was also considered. A study with 105 subjects revealed a variety of strategies and information about children's knowledge of commutativity and associativity. Four levels of performance of commutativity were also identified. A longitudinal study was carried out with 12 children in order to obtain details of children's changes in strategy, and to double check the results obtained in the main study. The strategies observed to be used by children over a 20 month period parallel those found in previous studies, which show a general transition to more efficient methods. However, the longitudinal study revealed that development of such arithmetic strategies is a slow process. Furthermore, the studies indicated that knowledge of commutativity is a prerequisite for associativity.

Models of the observed strategies have been implemented in the form of production rules in a computer program called PALM. The process of implementation highlighted features of children's problem solving that had not been detected during the studies.

In addition to models that describe the space of strategies, a model of learning has been implemented for the transition from procedural knowledge of commutativity to that of associativity. The model is capable of generalizing its inbuilt knowledge, for instance, its ability to solve 2-term arithmetic expressions, to allow it to solve more complex problems, such as 3-term arithmetic expressions. A further model has been constructed for learning arithmetic strategies that are more efficient than those already represented in the program. It learns specific rules by adding conditions for efficient problem solving to its previous general rules.

Dedication

To the memory of my father who died on 26th January, 1991.

Acknowledgements

I would like to thank my supervisors Mark Elsom-Cook, Tim O'Shea and Sara Hennessy for their support, guidance and encouragement during the course of this work.

I am indebted to the following for their careful and thorough readings of drafts of the thesis: Sara Hennessy, Mark Elsom-Cook, Tim O'Shea, Rick Evertsz, Maria Yannissi, Ronnie Singer, Fiona Spensley, Iraklis Paraskakis and Ihsan Al-Sabri. I would also like to thank Diana Laurillard, Claire O'Malley and Ann Floyd for their readings, criticisms and ideas on the empirical chapter of the thesis.

I would like to thank the head teacher of Saru M. G. M. school in Fiji, Mr Prasad, for giving me access to the children at his school, and Ms Jones of Simpson County Combined school (Milton Keynes) for her interest and help during the course of my longitudinal study. I would also like to thank the children who participated in the studies.

I thank Rick for his help and expertise with the Xerox machine and for his advice at the early stages of this research.

Thanks to everyone in CITE for providing a wonderful working environment.

Thanks to Rae, Laurence, Maria, Sonia, Kate, Pat, Iraklis, Jaya, Donald, Ronnie and Richard for their friendship; Ihsan for sharing this experience with me and for providing his shoulder to weep on; my parents, brother and sisters for their love and support.

Finally, I thank Markus Lusti for being sympathetic and for being flexible on the starting date of my job.

This research was supported by Rank Xerox and the Open University.

Table of Contents

Chapter 1

INTRODUCTION

The research reported in this thesis was undertaken in the context of Intelligent Tutoring Systems (ITS). The general goal of the research is to improve on the 'snapshot' approach to student modelling by modelling learning processes. This objective required two specific and related activities, constructing computational models and carrying out empirical observations of students' performance. Such computational models can form the student modelling component of an ITS. In order to construct such models, empirical studies were carried out to improve our understanding of children's behaviour in the particular domain.

This chapter provides an introduction to the research presented in the thesis. It begins with the objectives of the research. Next the motivations for choosing the research problem are discussed. Then the methodology of the research is described in detail. This consists of empirical work, production-rule modelling and computational models of learning. The last section of the chapter presents an outline of the rest of the thesis.

1.1 Objectives

Refinement of our understanding of children's cognitive processes can make significant contributions to educational and ITS research. The objective of the research described in this thesis is to understand children's performance on problems related to the concepts of commutativity[1] and

---

[1] Commutativity of addition: the order in which two numbers are added does not make a difference to their sum (a + b = b + a for all real values of a and b).

associativity[1] through empirical studies, and computer simulations of the observed performance on specific tasks.

Another research goal is to understand the developmental or learning process. The second objective of this thesis concerns this general goal. The objective is to model learning in order to predict the mechanisms for transition from one 'snapshot' model to another. A 'snapshot' model is one that describes a situation at one particular instance in time. It is static, that is, it does not change over time. The 'snapshot' modelling approach in an ITS implies that either a student's cognitive processes do not change over time or that if they do, then they 'magically' transform into the next 'snapshot'. It does not attempt to explain this 'magical' transition phase. For similar reasons, it assumes that bugs are stable. If there is any instability, there is no mechanism that can take it into account. Dynamic modelling, with plausible learning mechanisms, has the potential for modelling a student as his/her cognitive processes change during the course of interaction with the ITS.

1.2 Background and Motivation

1.2.1 Student modelling and models of learning

An ITS is a computerized teaching program which normally consists of the following components:

i) domain knowledge,

ii) student model,

iii) tutoring component and

---

[1] Associativity of addition: the order in which three numbers are added does not make a difference to their sum.

iv) user interface.

The above framework, due to Hartley (1973), has highlighted the importance of the student modelling component. Self (1974) emphasised that any computer-aided instruction program must maintain a student model. He also proposed what such a model could look like. Despite the computational limitations, Self (1988) continues to argue that it is a fundamental component of an ITS. Student modelling forms the background to the research presented in this thesis. There are two main approaches to student modelling. The first is by representing the student's knowledge as a *subset* of an expert's. This approach is termed *subset* or *overlay* modelling. An example of this kind of modelling is Young and O'Shea's (1981) accounts of children's performance on subtraction problems using 'rule deletions'. The second approach, referred to as *differential* or *perturbation* modelling, represents the student's knowledge as *incorrect* and *different* from an expert's knowledge. This is normally done using a library of *bugs* or *malrules*, where a bug or a malrule represents an erroneous version of a correct rule. Some systems that incorporate modelling of errors as modified versions of correct rules include LMS (Sleeman and Smith, 1981) and BUGGY (Brown and Burton, 1978).

It is not easy or necessarily possible to represent completely the student's knowledge with either approach. The subset approach does not take into account the student's conception/representation of the domain, which may not necessarily be the same as the expert's. The perturbation approach requires extensive domain analysis in order to create a library of bugs. It also has the problem of not taking into account the instability of bugs. In addition, neither of them takes into account how bugs are acquired. An incorrect response on a problem solving task may be a result of incomplete knowledge and/or incorrect versions of the target knowledge and/or lack of understanding of the basic concepts involved. It is crucial to know the semantics behind the malrules generated by the student and the processes

by which errors arise. An ideal model of a student would contain his/her current knowledge state, which may consist of malrules, and an explanation of how the student reached that particular state.

The limitations of the subset and of the perturbation modelling approach, and the target of the ideal model have been widely recognized (e.g. Brown and Burton, 1978; Clancey, 1986; Devi, 1989, in press; Elsom-Cook, 1984; Evertsz and Elsom-Cook, 1990; Hennessy, 1990; Laurillard, 1990; Payne and Squibb, 1988; Plotzner et al., 1990; Self and Gilmore, 1988; Spada et al., 1989; Wenger, 1987). More recently, attempts have been made at *generating* models that represent buggy behaviour, which were not accounted for by the particular system's prestored library of malrules. Some such attempts include repair theory (Brown and VanLehn, 1980), PIXIE (Sleeman and Smith, 1981) and ACM (Langley, Ohlsson and Sage, 1984). These and other work on computational models of learning are reviewed in chapter 2.

One approach towards constructing better student models is dynamic modelling. This involves modelling the student's learning process, which not only identifies his/her knowledge at any one time, but also identifies the way in which that knowledge is acquired. The need for research concerning such dynamic modelling was established as early as 1976. Young (1976) modelled children's performance on seriation tasks at different stages of development using production systems. He concluded that production systems had served as a medium in which development could be analysed and discussed. He emphasised that the real problem was to build a production system which itself develops.

The genetic graph (Goldstein, 1982) representation of student models in WUSOR-III has been the only attempt at modelling evolution of knowledge in an ITS. The knowledge is represented using genetic graphs. The rules form the nodes of the graph which are interlinked by relations such as generalization, analogy and refinement. The links denote the 'learning' relationship between rules. The idea of representing different learning

strategies using genetic graphs is an attractive one. By tracing paths on the graph, it has the potential of explaining a student's development of knowledge in terms of the procedural rules and the learning strategies. As it is, the genetic graph represents a student's learning as an overlay of an expert's. In addition, it is static, that is, it is predetermined, and once it is programmed, it remains unchanged. If the genetic graph idea of representing knowledge and learning strategies could be dynamic, then it would have great potentials for an ITS. Dynamic modelling would enable an ITS to make more informed predictions about its teaching strategies. For example, the system could hold knowledge about teaching strategies associated with learning strategies. Once the system has the ability to predict the student's learning strategy, it can use this information in addition to his/her current knowledge state to choose its teaching strategy. The tutor can also reason about what skills the student is ready to acquire.

There are a number of existing theories of procedural skill acquisition (e.g. Anderson, 1983, 1986, 1987; Brown and VanLehn, 1980; Ohlsson and Rees, 1988; Rosenbloom and Newell, 1986; VanLehn, 1983). Rosenbloom and Newell (SOAR) focus on chunking of knowledge. Anderson's ACT* has knowledge compilation as one of its learning features. The objectives of both SOAR and ACT* are to model the increase in efficiency of a system with practice. VanLehn (SIERRA) concentrates on induction and knowledge integration from a sequence of lessons. Ohlsson and Rees (State Constraint Theory) model learning as a result of violations of conceptual knowledge. A more detailed review of these theories of skill acquisition is presented in chapter 2. The learning model presented in this thesis draws on previous models of learning and is applied to the specific domain of elementary addition. Two types of learning are considered here. Firstly, failure-driven learning, which involves the application of existing knowledge to a new situation. This is achieved by using a learning by generalization technique (Michalski, 1983). Secondly, efficiency-driven learning, which considers the

application of existing knowledge for learning more efficient problem-solving strategies.

1.2.2 Commutativity and associativity

In order to create a model of learning, a task in a real domain that could be studied empirically, needed to be chosen. The domain of mathematics was chosen since most of the computational modelling research has been carried out in this domain and hence is ideal for comparison. Some examples of such computational models are Brown and VanLehn (1980), Langley, Ohlsson and Sage (1984) and Young and O'Shea (1981) for the subtraction domain, Sleeman and Smith (1981) for algebra, Greeno, Riley and Gelman (1984) and Ohlsson and Rees (1988) for counting and Neches (1987) for addition. Details of such systems are presented in the literature review in chapter 2. The specific aspects of the modelling in these systems which can be compared are the type of the domain, the representation of knowledge, the proportion of a sample of subjects' performance that they are capable of taking into account, and the generality of the system's modelling approach.

Within the domain of mathematics, a specific area needed to be chosen which was complex enough for a detailed model, and which involved a transition that could be studied empirically. The arithmetic concepts of commutativity and associativity were chosen for several reasons. Firstly, these are important basic concepts, which reduce the amount of number facts that children need to remember. For example, if a child knows 4 + 2 as a number fact, and knows the concept of commutativity, then s/he does not need to store 2 + 4 as another number fact. Besides this, the concepts form the foundation for further arithmetic development. This includes invention of informal algorithms, transition to more efficient strategies, and the application of the concepts to algebra in general.

The second reason is that the concepts are good examples for studying conceptual and procedural knowledge. Previous related research provides conflicting views on whether the ability to execute a procedure necessarily implies underlying conceptual knowledge, and whether the presence of conceptual knowledge necessarily means that it will be applied (e.g. Baroody and Gannon, 1984; Briars and Sieglar, 1984; Fuson, Secada and Hall, 1983; Gelman and Gallistel, 1978; Gelman and Meck, 1983, 1986; Greeno, Riley and Gelman, 1984). Experiments by these researchers leading to their arguments are described in the review chapter. Detailed studies could provide information to clarify the relationship between conceptual and procedural knowledge. Thirdly, little research exists on these basic concepts in arithmetic. The concepts of commutativity and associativity are closely related concepts but there has not been any investigation of the transition from commutativity to associativity. On the other hand, the acquisition of the concepts follows the acquisition of counting skills for which there exists a substantial amount of research, and this provides useful background information.

The investigation of the concepts of commutativity and associativity is also interesting because it is debatable whether they should be taught. It has not been agreed by the education community whether the concepts should be explicitly taught, or whether they should be left for the students to discover themselves. Moreover, it is a domain in which children's specialization and generalization can be studied. For example, children generalize commutativity to subtraction, that is 4 - 5 = 5 - 4. Examples of children's specialization of the concept of commutativity include its application to small numbers only, or in concrete cases only. Finally, findings from the chosen task can be tested for generality to other related tasks, for example, fractions and algebra.

In sum, there has not been any empirical work which indicates how the transition from a child's concept of commutativity to that of associativity

takes place. Furthermore, there has not been any cognitive modelling of this transition. This research uses empirical evidence and techniques from machine learning and theories of skill acquisition to model a possible mechanism for the transition.

1.3 Research methodology

The methodology employed here follows on from that used in Devi (1990b) and Devi et al. (in press). It can be summarised as follows:

empirical studies --> models of performance --> model of learning

The models of performance are based on empirical work. The model of learning requires a model of performance of the initial state, from which to begin learning. It also requires empirical observations of the initial state, the goal state and of the transition from one state to the other. The process of empirical observations and computer models of observed performance can be seen as iterative. The computational models provide feedback for the empirical analysis, which in turn provides feedback to the former. Ideally, the results of the learning model can be used to design further empirical tasks, which in turn could provide more 'fine-tuned' analysis to improve the learning model.

In order to construct models of performance, some understanding of children's learning in the particular domain is required. Three studies (pilot, main and longitudinal) were carried out to examine children's strategies and their understanding of the concepts of commutativity and associativity. Children were observed solving elementary arithmetic problems, like 5 + 8 and 6 + 5 + 3. They were interviewed and their verbal think-aloud protocols were tape-recorded. The pilot study gave an indication of the age range that should be studied and the tasks that could be performed in order to draw the most out of the students. The main study

provided a space of strategies that children at different levels used. It also proposed performance levels of the concept of commutativity. Furthermore, the main study investigated the transition from commutativity to associativity. The longitudinal study was carried out to investigate the developmental aspect of learning. It was concerned with observing children through a sequence of levels. This study provided more detailed analysis of children's learning, for example, change in strategies over time. It also confirmed the results obtained in the main study.

Following the pilot and the main studies, and in parallel with the longitudinal study, computational models of children's observed strategies were constructed. A production-rule modelling approach was used for constructing models of children at different levels of development. The models are designed to capture the child's state at a particular time. They are equivalent to 'snapshot' models described earlier. They do not represent a continuous picture. The models at each level represent only certain aspects of a model of learning.

In their information-processing theory of human problem solving, Newell and Simon (1972) propose that computer simulations can produce behaviour that closely resembles human behaviour in the same problem-solving situations (Simon, 1985). The research described in this thesis makes the assumption that human cognitive processes can be modelled as production systems. Our choice of the production-rule modelling approach has been influenced by Brown and VanLehn (1980), Klahr and Wallace (1976), Langley, Ohlsson and Sage (1984), Ohlsson and Rees (1988) and Young and O'Shea (1981), who have demonstrated that some aspects of arithmetic skills can be usefully modelled using production rule systems. Some of these modelling approaches are reviewed in the next chapter. Furthermore, such models can in turn be used for student modelling in ITS like those developed by Anderson, Boyle and Yost (1985), Anderson and Reiser (1985), Clancey (1982) and O'Shea (1979). Hence, one of the reasons

for choosing the production-rule formalism is its application to student modelling. A further reason for using this formalism is the modularity and extensibility of production systems which allow mechanisms which make them learn to be incorporated.

Learning models which describe children's transition from one performance level to another, and which model the transition to more efficient strategies are presented in this thesis. Learning in the models is initiated by one of two reasons, failure or efficiency.

A model of failure-driven learning that describes a possible mechanism for the transition from the procedural knowledge of commutativity to that of associativity is presented. It occurs when there are no rules that are applicable to the current problem solving state. The model learns by generalizing its existing rules. The second model, efficiency-driven learning, is based on ACM-like operator applicability (Langley, Ohlsson and Sage, 1984). The system calculates estimates of efficiencies of strategies, and its goal is to learn strategies that are more efficient than those it already knows. It learns more efficient strategies by adding known facts as specific conditions to existing more general rules. In the case of arithmetic, the known facts are number facts like 5 + 5 = 10. The efficiency-driven model of learning learns conditions where such number facts could be applied for solving problems in order to save the effort in computing it.

1.4 Outline of thesis

The thesis contains six chapters. It reports the development of a candidate mechanism for the transition from commutativity to associativity. A computer program, PALM (Production-rule Arithmetic Learning Modeller) is used to simulate children's problem-solving strategies related to the concepts.

Chapter one (this chapter) provides an introduction to the thesis. It describes the objectives, the motivations, and the methodology employed for the research presented in the thesis.

Chapter two is a review of the related research. Since the research methodology is twofold: *empirical investigation* of commutativity and associativity, and *computational models* related to the concepts, the literature reviewed in this chapter consists of these two areas of research. The first section of the chapter is a review of empirical investigations of concepts and procedures concerning the arithmetic principles, commutativity and associativity. This section relates to chapter three. The second section, which relates to chapters four and five, reviews some of the computational models that have been constructed for the domain of arithmetic.

Chapter three is an account of the empirical studies that were carried out to investigate children's acquisition of commutativity and associativity. The studies consisted of a pilot study carried out with 22 children, a main study carried out with 105 children, and a longitudinal study carried out with 12 children. In the studies, children's strategies and their underlying conceptual knowledge were investigated. The chapter also discusses the performance levels of commutativity that were identified in the studies, and proposes some explanations for children's answers to subtraction problems, including generalization. Observations from the studies were used for constructing production-rule models.

Chapter four presents a production-rule account of children's strategies on problems related to commutativity and associativity. The simulations of the observed strategies and their efficiencies are described. The models represent 'snapshots' of children's problem solving behaviour. As in previous production system models, the models implemented in PALM do not describe the processes involved in development.

Chapter five describes an extension of PALM to model a possible mechanism of learning. The learning model is intended to facilitate the construction of valid models of children's performance. The program includes two types of learning: failure-driven learning and efficiency-driven learning. Failure-driven learning occurs when there are no applicable rules to solve a given problem. The program learns by generalizing its existing rules. Efficiency-driven learning is employed to learn more efficient strategies than those that the system already knows.

Chapter six presents a summary of the thesis. It highlights the contributions of the research and discusses the implications of the modelling for ITS. Some directions for further research related to empirical work and computational modelling are also proposed.

Appendices 1 to 6 give some additional information on the tasks and the results of the empirical studies presented in chapter 3. These include examples of the types of problems that were presented to the students, a sample protocol of an interview, and the details of each child's performance in the main study.

Appendix 7 is a listing of the program.

Chapter 2

RELATED RESEARCH

As outlined in the first chapter, the approach to this research involved empirical studies of children learning arithmetic so as to understand their cognitive processes. Following this, computational models of pupil competence at different stages of development were built in order to clarify what is being learnt and why pupil behaviour is changing.

This chapter is a review of related research on children's acquisition of arithmetic skills. It concentrates on the two methodologies that are employed in the research presented in this thesis: empirical studies and computer simulations. The chapter is divided into two main sections. The first section discusses empirical investigations of children's learning of arithmetic skills. This includes children's competence in counting and the development of their ability to add. The second section is a review of computational models of children's arithmetic performance and skill acquisition.

2.1 Acquisition of arithmetic skills

2.1.1 Introduction

There is a debate in the literature about the status of conceptual and procedural knowledge. These terms are not that precise, but in this thesis we will use them as defined by Hiebert and Lefevre (1986). Conceptual knowledge consists of facts and their relationships. The term 'understanding' is used interchangeably with conceptual knowledge by some researchers. Procedural knowledge consists of rules, algorithms and strategies for carrying out tasks, or for solving problems. A skill is the

ability to execute a procedure. It may be either totally procedural or it may have associated conceptual knowledge. The current literature provides evidence of the view that procedural knowledge (of early arithmetic and counting) is acquired before conceptual knowledge (Baroody and Gannon, 1984; Briars and Sieglar, 1984; Fuson, Secada and Hall, 1983), as well as the view that conceptual knowledge is acquired first (Gelman and Gallistel, 1978, Gelman and Meck, 1983, 1986; Greeno, Riley and Gelman, 1984). In this section we review work done by these researchers to investigate children's knowledge of arithmetic skills and concepts. First, we review empirical investigations of preschool arithmetic knowledge. This is followed by a review of work on addition strategies and the principle of commutativity. The last section is a review of research on the representation of number knowledge.

2.1.2 Preschool knowledge and counting

Gelman (1972, 1977, 1982), who has done much research into preschool children's development of arithmetic reasoning principles, holds the view that principles are acquired before skills. She maintains that concepts and principles are used in constructing or acquiring procedures. In order to find out how children reason about small numbers, Gelman carried out a series of "magic" experiments. In the experiments, children were shown two plates containing different numbers of plastic toys. One of the plates was designated "the winner" without the children being told why. Then the children were asked to identify the winner, and to justify the properties of the two sets that they used to get their answers. Gelman found that the children almost always used numerosity to distinguish the two sets, for example, "Plate 1 wins because it has three ...". From her experiments, Gelman concluded that children as young as three years possess arithmetic reasoning principles. Some of these principles are:

1) *Equality*: Children are able to identify two sets with equal numerosity. The equality relation is one of the several basic pieces of knowledge that is required later on in children's understanding of the commutativity principle.

2) *Order*: This relation follows from the previous one. Gelman (1977) provides evidence that when children recognize that two sets are not equal, they know that an ordering relation exists between them, i.e. that one set is more than the other.

3) Addition increases numerosity and subtraction decreases numerosity.

4) *Solvability principle*: Addition is the reverse operation to subtraction.

5) *One-one principle*: Each item in the set being counted is assigned one (and only one) tag.

6) *Stable-ordering principle*: The tags used in a count must be arranged in a stable order (e.g. 1, 2, 3 and not 2, 1, 3).

7) *Cardinal principle*: The last tag used in the count of a set represents the number of items in the set.

8) *Abstraction principle*: The counting procedure can be applied to any collection of objects.

9) *Order-irrelevance principle*: While assigning tags to objects in the set, it does not matter which tag is assigned to which object. For example, while counting a set of two fruits (a banana and an apple), it does not matter whether the apple or the banana is assigned the tag '1'.

Principles 5, 6 and 7 together make up the prerequisites for the ability to count. Gelman and Gallistel (1978) provide evidence that even some two-year-olds know these three principles. More than 90% of their four- and

five-year-olds and 80% of the three-year-olds honoured the stable-order principle. To test the order-irrelevance principle, they had the children count the same set several times, each time tagging a different object as one. The results of this task showed that most five-year-olds had explicit knowledge of the principle. Gelman and her colleagues (Gelman and Gallistel, 1978; Gelman and Meck, 1983) propose that preschoolers' counting is governed by the implicit knowledge of the counting principles. Implicit and explicit knowledge of the principles is distinguished by the ability to verbalize or state the counting principles and the ability to demonstrate that one's behaviour is systematically governed by the principles. They support their conclusion by their experiments to assess a child's ability to detect errors in a puppet's application of the one-one, stable-order and cardinal count principles. Note that Gelman and Gallistel infer this from children's performance. From children's performance, we cannot conclude for certain that they have the underlying principled knowledge. Their knowledge might be limited to the ability to execute a procedure only. In addition, even if experimental results show the presence of conceptual knowledge, it does not imply that conceptual knowledge is acquired before procedural knowledge.

Briars and Siegler (1984) carried out experiments to investigate whether children knew the principles underlying their counting procedures. Preschoolers' knowledge of counting principles was investigated by examining their ability to discriminate between features that are essential for correct counting and features that are typically present but unessential. Three to five-year-olds were asked to judge a puppet's counting as either acceptable or not acceptable. Each child's skill at counting rows of objects was also assessed. They found children who could count correctly but could not consistently judge the puppet's counting errors as incorrect. This led to Briars and Siegler's conclusion that skill in executing the standard counting procedure was found to precede knowledge of the underlying principle. Note that Briars and Siegler assumed that if a child could

identify another individual's errors, then it implied that s/he had the principled knowledge. In addition, they did not find any child in their experiments who consistently detected the puppet's counting errors and failed to count correctly themselves. From these findings, Briars and Siegler hypothesized that it is improbable that knowledge of principles guides acquisition of counting procedures.

The debate on the interaction between conceptual and procedural knowledge as children learn to count, remains unresolved. Gelman and Gallistel (1978) and Gelman and Meck (1983) have proposed that preschoolers counting is directed by the implicit knowledge of the above five counting principles. Briars and Siegler (1984) and Fuson, Secada and Hall (1983), on the other hand, believe that children initially display various counting behaviours without understanding and only eventually induce principles or components of the counting principles. More recently, Gelman and her colleagues seem to have moderated their position in this debate. Gelman, Meck and Merkin (1986) proposed that conceptual competence does not provide recipes for procedures but it does set constraints on the class of procedures that procedural competence can generate. Referring to the case of counting, they conclude that conceptual competence can guide the acquisition of skill. They also acknowledge that, alternatively one could argue that there are cases where conceptual competence (a principle) develops out of procedural competence (a practice). They even provide an illustration of how procedural competence can lead to the acquisition of conceptual competence. They conclude that conceptual competence need not be entirely or even mostly preformed, but a preformed kernel is a prerequisite for the development of both procedural competence and further conceptual competence.

2.1.3 Addition strategies

Concrete understanding of addition

Gelman and Gallistel, in their "magic" experiments, showed children a set of objects on a plate, and then showed them a plate with a different number of objects. They found that children as young as three could distinguish between the plates, and could even explain the transformation that had taken place: for example, "you took one off", "you put one on". This shows that these children have formed a notion of addition and subtraction in terms of the concrete actions of 'adding on' and 'taking away'.

Starkey and Gelman (1982) conducted a study which provides evidence of young children's use of counting algorithms. They conducted a study to determine whether young preschool children were capable of solving a variety of problems where objects were added to or subtracted from a set of objects that were screened from the subject's view. Each of the tasks began by having the children establish the number of pennies held in the experimenter's open hand. The experimenter then screened the set of pennies and placed another set in the same hand, stating the number of pennies he was putting. The subjects were asked how many pennies the experimenter had in his hand. The pennies were covered so that the subjects could not count them to get the answers. The experimenters found that preschool children (especially four- and five-year-olds) were still able to use counting algorithms in this situation. Some children used fingers to represent the screened objects; others counted aloud. The results of the study indicate that some preschool children (including a few three-year-olds) can use counting algorithms even when the set of objects are screened from view.

Hughes (1981) carried out an experiment where the task was very similar to that used by Starkey and Gelman, except that the problem concerned bricks

in a box. He obtained similar results to Starkey and Gelman; he found that young children were able to solve simple addition (and subtraction) problems even when no visual apparatus was present.

Thus there is evidence from the above studies that many preschoolers are able to solve simple arithmetic problems in a concrete setting. Solving problems with concrete objects is the earliest stage in addition and subtraction. The most common addition strategy at this stage is 'counting all', where concrete objects are counted to represent each addend, and then they are all counted to get the sum (Groen and Parkman, 1972; Ilg and Ames, 1951). The next stage is the abstraction of the previous concrete stage, where representation of the problems with concrete objects and strategies like 'count all' are replaced by more efficient strategies like 'count on'. The details of the strategies and their transformations are discussed in the following section.

Commutativity and development of strategies

One of the studies conducted by Baroody, Ginsburg and Waxman (1983) included the use of the commutativity principle (by children aged from five years ten months to nine years). Each child was tested individually with a sequence of ten problems like 13 + 6, 6 + 13, 14 + 7, 7 + 14, in the context of a game. The game was called Math Baseball. The length of time taken by the child to solve each problem was noted. If the child was correct, then his/her batter (a block 2cm x 0.8cm x 0.8cm) got a hit and could move to the first base. The batter could move two, three or four bases, depending on how long the child took to solve the problem. The objective of the game was to get the children to solve the problems in the quickest way, and this was explained to the subjects. It was found that most of the children used the commutativity principle. This was noted by observing whether or not the child looked at the previous problems and by noting other actions like finger counting. About three quarters of the children used the principle on the

first pair of examples. Most of the second and third graders (approximately 85%) used the principle consistently. The younger ones did not use it as consistently (59% were consistent users). The authors propose that this might be due to the fact that younger children do not necessarily appreciate that the principle extends to large numbers.

It is not yet clear how the principle develops. The children who were studied by Baroody, Ginsburg and Waxman had no formal instruction concerning the commutativity principle. Hence, they proposed that its development is probably the result of informal experience. For example, when dealing with concrete objects, most children do not pay any particular attention to the order of the addends. Even while doing mental addition, children often disregard addend order. A commonly observed strategy is starting from the larger of the addends and counting on the other addend. For example 4 + 7 would be solved as - start from 7 and then count on 4 - 8, 9, 10, 11. It has been assumed that this strategy is an application of the commutativity principle. However, this may not always be true, because it is also possible that children use this strategy to save mental labour, without any knowledge of commutativity. The knowledge of principles and the ability to solve problems that could be solved using the principles need to be distinguished.

Baroody and Gannon (1984) provide evidence of children who use the above strategy of disregarding addend order in solving addition problems, and yet do not succeed in commutativity tasks. Of the fourteen subjects who used the more advanced strategy of counting on from the larger addend, 57% were successful on the commutativity tasks, 21% were inconsistent, and 21% unsuccessful. From this, Baroody and Gannon correctly concluded that a child who disregards addend order, does not necessarily appreciate commutativity. One such child in their study said that the sum of 7 + 2 and that of 2 + 7 were "different". Then the child was asked to compute 7 + 2. After he counted and responded correctly, he was asked how much 2 + 7

was. The child was completely unaware of the similarity of the problems and counted to give the answer. Another child, after she had computed 6 + 4, was asked if 4 + 6 would produce 10 - the same or different answer as 6 + 4. She thought for about a minute, computed the sum, and then said "the same". From these two children's invention of 'count on from the larger addend' (COL) strategy, one can conclude that the appreciation of commutativity is not necessary for the invention of labour-saving addition strategies.

Resnick and Groen (1977) conducted a study in which preschool children were taught the 'count all' strategy of addition (count out each of the addends, combine the two sets and then recount). After several practice sessions (over several weeks), half the children had switched to the 'count on from the larger addend' strategy without being taught. This result suggests that strategies like 'count on from the larger addend' are acquired without instruction and are invented by children. Similarly, the concept of commutativity might be acquired without instruction.

Children's counting strategies for addition normally develop from 'count all' to 'count on from the first addend' to 'count on from the larger addend' (Baroody and Gannon, 1984; Gelman, 1977; Resnick, 1980). In the 'count all' strategy, fingers or physical objects like unifix cubes are used to count out each of the addends, and then the two sets are combined and recounted. In the 'count on from the first addend' strategy, the counting begins from the first addend, and not from '1' as in the previous strategy. In the 'count on from the larger addend' strategy, the counting begins from the larger of the addends. This strategy is usually referred to as 'min' by other researchers like Groen and Parkman (1972) and Resnick (1980). It is quite logical to believe that this strategy follows from the commutativity principle, but as discussed above, Baroody provides evidence of this not necessarily being the case. Furthermore, Resnick and Groen's (1977) study provides reasons to question the assumption that the use of 'count on from larger addend'

strategy implies some knowledge of commutativity. This is because in their experiment, the children invented the strategy under controlled practice conditions, in which commuted pairs of problems (e.g. 6 + 8 and 8 + 6) did not occur. It seems likely that after practice at solving many addition problems, children start using the 'count on from larger addend' strategy as a result of the search (perhaps unconsciously) for a solution with minimum effort. Resnick (1983) provides the following as a possible explanation for children's use of the 'min' strategy:

> "Since 'min' works (i.e. the answer turns out to be correct when checked by counting the whole joint set, and adults do not comment on the result as wrong), they would retain it as a perfect procedure." (Resnick, 1983, p. 123).

She further explains that a natural extension of the order-irrelevance principle would allow the count on from the larger addend strategy to emerge as part of a general search for low-effort solutions without requiring that the child construct any kind of commutativity rule. This is consistent with Baroody's conclusion that the invention of the 'count on from the larger addend' strategy to minimize mental computational effort does not necessarily imply that the child appreciates commutativity. It implies or requires only protocommutativity[1] or perhaps just an order-indifferent[2] tagging scheme. Furthermore, the heuristic procedure modification program (Neches et al., 1987; described in section 2.2.2), which simulates children's counting strategies, demonstrates the transition to 'counting on from the larger addend' strategy based only on the motivation of reducing the amount of work required, and not on the principle of commutativity. The program does not assume commutativity but does

---

[1] Protocommutativity: the order in which addends are dealt with does not make a difference in terms of the correctness of the sum (Baroody et al., 1984).

[2] Order-indifferent tagging scheme: elements of a set may be enumerated in any order.

assume an order-indifferent tagging scheme for counting (Baroody and Gannon, 1984).

Besides the use of counting strategies, children rely heavily on known number facts in their solutions to arithmetic problems. It has been noted by several researchers (e.g. Carpenter and Moser, 1983) that some number facts (e.g. doubles) are known to students at an early age. For example, 8 + 9 = ? may be solved as: 8 + 8 = 16 and 1 = 17. Resnick found that children who were using counting strategies, did not use them for "doubles" problems like 2 + 2, 3 + 3. Instead these problems were solved very quickly - the answers were probably already in long-term memory and were recalled directly (Resnick, 1980).

2.1.4 Representation of number knowledge

One of the common means used by cognitive scientists for representing knowledge is in the form of semantic networks. Resnick and Ford (1984) provide an analysis of number understanding based on this representation of structuring and organizing information in the long term memory. In the semantic network representation, concepts are represented hierarchically with links between them and their properties. Because of the links, the interrelationships between the pieces of knowledge can be easily represented. In order to retrieve information, and to generate information that can be used directly, Resnick and Ford take the information-processing view that in addition to knowledge structures, the brain possesses a repertoire of problem solving strategies that help to interpret problems, locate stored knowledge and procedures, and generate new relations among separately stored memory items.

Resnick's (1983) account of the part-whole representation of children's number concepts is based on the semantic network representation. She proposes that with the application of part-whole schema to quantity,

children begin to think about numbers as compositions of other numbers. The part-whole schema specifies that any quantity (the whole) can be partitioned (into the parts) as long as the combined parts neither exceed nor fall short of the whole. Partitioning numbers into their parts is a common step in children's informal algorithms. However, the part-whole schema representation of knowledge is only a possibility. There is no evidence to show that the children who use partitioning in their invented algorithms make use of this mental representation.

Baroody (1985) argues that mental representation and efficient recall of number combinations may be more elaborate than simple associative networks. Because people are flexible information processors, they may use several means to generate number combinations - including reconstructive processes (Baroody and Ginsburg, 1982). He proposes that rules, procedures and principles are stored in memory and number combinations are generated using them. In making use of these rules, principles and procedures, one does not need to learn and store all the individual number facts or combinations.

A more complete model of representation of knowledge would be one which includes both the reproductive process (information is stored as facts e.g. 2 + 2 = 4, and retrieved without the need to use any procedures), and the reconstructive process (number combinations are generated using stored procedures, principles or rules). Number facts like 4 + 1, 5 + 1, 1 + 3, are more probably stored as rules, e.g. 'adding 1 to any number gives the next higher number', than as facts like 1 + 1 = 2, 2 + 1 = 3, 3 + 1 = 4, etc. On the other hand, number facts like 3 + 4 = 7, 2 + 8 = 10, are more probably stored as facts, as a result of having seen and done such problems several times, rather than as procedures that generate the sums. Very familiar combinations with great associative strengths might always be retrieved from a factual representation. Unfamiliar combinations would probably be generated from the representation of the rule (Baroody, 1985).

2.1.5 Summary

The above review of counting and of commutativity and addition strategies demonstrates that the development of strategies or procedural knowledge is not necessarily dependent on the presumed conceptual knowledge. Certainly children can learn procedures by rote without relating them to any appropriate form of conceptual knowledge, and some invention appears to occur strictly within the context of procedural knowledge (Brown and Burton, 1978; Brown and VanLehn, 1982). Gelman and Meck (1986) explain the fact that children vary in their ability to succeed on counting tasks by both a principle-first and principle-after account of early numerical skill. Baroody and Ginsburg (1986) and Silver (1986) argue that in many cases the development of conceptual knowledge is neither necessary nor sufficient to ensure the acquisition of related procedures. Baroody and Ginsburg propose that the use of advanced strategies is as much a result of reducing cognitive processing demands as it is of the acquisition of underlying conceptual knowledge. In fact, one could argue that reduction of cognitive effort has to be there to generate the change. The application of certain procedures may lead to conceptual knowledge rather than vice versa, as children note regularities in applying the procedures (Carpenter, 1986; Baroody and Ginsburg, 1986). Gelman and Meck (1986) provide a view consistent with Carpenter and Baroody and Ginsburg: procedural competence can lead to the development of new principles of conceptual competence.

The review above demonstrates the need to make a distinction between conceptual knowledge and procedural knowledge. Work to date has not established the relationship between the two types of knowledge. For example, it is not known whether one of them is acquired before the other, or whether they are acquired simultaneously. From what is known so far, it can be concluded that the important thing is that for a better understanding, children should possess both, and know the relationship

between them. In order to facilitate children's understanding of their arithmetic procedures, instruction should focus on the link between procedures and their conceptual knowledge. The need for such a link has been emphasised by other researchers (Carpenter, 1986; Gelman and Meck, 1986; Hennessy, 1986; Resnick, 1980). In addition, one type of knowledge would most probably facilitate the other. Just as procedures occasionally generate or advance concepts in mathematics, new procedures can trigger for individuals the development of concepts (Hiebert and Lefevre, 1986).

In summary, the review above highlights the need to consider and distinguish between procedural and conceptual arithmetic knowledge. In chapter 3, empirical studies carried out to investigate children's knowledge of the concepts of commutativity and associativity further highlight the need to make a distinction between the two types of knowledge. The review of previous work on children's arithmetic reveals that there has been some research done on children's concept of commutativity. However, there is virtually no research on the concept of associativity. There is no research on the interrelationship between the two concepts. In the research reported in this thesis, empirical work was carried out to study the two concepts and their interrelationship.

2.2 Computational models of performance

Computational modelling, like carrying out empirical studies, is a means to investigate human performance. It is a methodology for understanding certain aspects of children's behaviour in a given domain. Moreover, the models can be used for student modelling in ITS. Research in cognitive modelling in the domain of arithmetic has concentrated on modelling children's errors as 'buggy' procedures. More recently, attempts have been made to create models of learning and development. In the following section, some such approaches to modelling are reviewed.

2.2.1 Planning nets

Greeno, Riley and Gelman (1984) developed a model of performance in counting tasks, called SC, for Simulation of Counting. They attempted to integrate the understanding of counting principles and performance in counting. As with other simulation models, in SC the principles remain implicit. The authors state that the principles are not represented directly, and the model is proposed as a hypothesis about children's implicit understanding of the principles. They use a 'planning nets'[1] formalism to make an explicit connection between hypotheses about conceptual competence and models of performance. The connection is made by assuming that performance is a consequence of conceptual competence. This is precisely the assumption that Ohlsson and Rees (1988) make in their model (discussed later). Conceptual competence is represented as a set of action schemata. These are equivalent to a set of axioms in the domain. For counting, the set of schemata correspond to cardinality, one-one correspondence and order principles. Representation of a schema includes its prerequisites (conditions of applicability), postrequisites (criteria for success), consequences (results of the schematic action) and corequisites (conditions during the execution of the action). The following is an example of the representation of a schema:

COUNT(X)

Prerequisites: set of numerals, N;
order(N).

---

[1] 'planning nets' are directed graphs. The nodes of the net represent plans, and the links represent planning inferences.

Postrequisites: equal(X,SN);
bound(SN,n).

Consequence: number(X) = n.

The schema describes the action of counting. The prerequisite for counting is an ordered set of N numerals (1, 2, 3,...). The action succeeds when X equals SN, where SN denotes an initial segment of N, and the upper bound of SN is n, which is the number of objects in the counted set. For example, to count 5 objects (X = 5), SN is the set of numerals, 1, 2, 3, 4, 5. The upper bound of SN is 5 (n = 5). Counting designates a numeral from SN to each object. The numeral designated to the last object, that is the upper bound of SN, represents the number of objects in the set.

The action schemata are used as premises for deriving planning nets for procedures, termed procedural competence. Procedural competence refers to knowledge of principles relating to goals, actions and the requisite conditions for actions. It uses heuristic planning rules, which recognize the different goals while planning, select the action schemata whose consequences match the recognized goals, set new goals based on the requisite conditions of the selected schemata, and determine when a plan is successfully completed. Connections between goals (procedural competence) and actions (conceptual competence) in the network correspond to relations that are explicitly stored in action schemata, such as consequences and requisites for actions. For example, the schema COUNT, expressed above, links COUNT(X) to NUMBER(X) because NUMBER(X) is a consequence of the COUNT action and COUNT(X) is connected to equal(X,SN) because equal(X,SN) is a requisite condition. This shows that for a connection between hypotheses about competence and a model of performance, one does not have to construct a planning net analysis - information represented in the action schemata seems to provide the relations.

Greeno, Riley and Gelman claim that the action schemata represent a plausible set of hypotheses about children's conceptual knowledge. However, they acknowledge that the way in which the planning nets are derived in order to make the connections between conceptual competence and performance, does not necessarily have any psychological plausibility.

2.2.2 HPM

Neches (1987) developed a computer program to model the development of addition strategies. The model includes improvements in the efficiencies of procedures. The system, called HPM (for Heuristic Procedure Modification) uses information on its past actions and its current set of rules to learn new rules.

Neches discusses the transition from CAF (count all starting from the first addend) to COL (start from the larger addend and increment it the smaller number of times) strategy using strategy transformation heuristics. For the CAF strategy, separate sets of objects are counted out to represent each addend and then combined into a single set, which is counted to find the sum of the two addends. While carrying out this procedure, some of the intermediate results include: 1) two sets of objects, with each object in each set having a number assignment, 2) the combined set has the same objects with their initial number assignments, but of unknown size since the objects have not been recounted, and 3) a set of known size, which contains the same set as in 2 above, except with their new number assignments. Part of the last step is redundant - a set of one of the addends in step 2 gets the same assignment in step 3. Eliminating the redundancy of recounting the objects representing the first addend produces a new procedure. In this procedure, while counting the combined set, the result of counting the first addend is used to initialize the counting of the second addend. At the end of this procedure change, it becomes possible to note that the counting of the first addend is not needed. The result of the count is the number

representing this addend, which is known in advance, and hence the number itself can be used. A more efficient procedure results by eliminating this inefficiency. This procedure counts the second addend only, and the count begins from the first addend (COF).

A more efficient procedure than the COF is COL/MIN. To go from COF to COL, HPM uses a heuristic called 'Effort difference', which is motivated by "try to use the method involving less effort".

HPM's knowledge of procedures is represented using a production-rule formalism. The condition sides of these rules match propositions in a goal structure and the actions add more propositions to the goal structure. The resulting goal structures are hierarchical graphs representing the execution of procedures. One of the nodes in the goal structures represents the effort involved in processing a goal. The effort of a process is defined as the size of goal structure underneath the goal node which initiates that process.

Some of the weaknesses of HPM concerning the simulation model not being psychologically valid are as follows:

i) The same procedure is applied for larger numbers as that for small numbers. Solving problems like 4 + 5 using counting is fine, but 36 + 74 would rarely be solved by counting.

ii) For the transition from COF to COL strategy, Neches says that HPM discovers effort differences between different trials with the same problems. There is no mention of knowledge of commutativity. It is assumed that 6 + 4 is the same problem as 4 + 6.

2.2.3 Repair Theory

Brown and VanLehn (1980) have proposed a theory of procedural problem solving, called Repair Theory. Given a procedural skill, the theory predicts the systematic errors or bugs that will occur in the performance of students learning that skill. The idea of children's bugs being systematic has been questioned and argued against by several researchers (Hennessy, 1990; Payne and Squibb, 1988). Apparent systematicity in children's performance may not always be based on stable errors.

Repair theory is based on the assumption that when a student gets stuck while solving a problem, that is, when s/he reaches an 'impasse', s/he attempts a 'repair' in order to get unstuck. It is assumed that the cause of an impasse is incomplete knowledge. Repair theory models an incomplete procedure by applying a set of deletion principles to a production-rule representation of the correct procedure. Filtering principles are used to restrict the deletion of those rules that generate 'core procedures' that are not plausible. Core procedures represent students' current, incomplete knowledge of the skill. The deletion principle is not meant to model children's acquisition of incomplete knowledge of procedures. It is used for generating impasses that require repairs. The set of repairs is defined by a set of 'repair heuristics', which propose repairs to impasses. Repair heuristics are instances of general problem solving heuristics like 'use an operation that worked in an analogous situation'. The repairs complete the core procedures, hence allowing them to proceed with the problem solving. A set of 'critics' is used to filter the set of repairs in order to avoid those that are implausible.

Repair theory does not take adequate account of the semantics of the procedures being executed; it does not take principled knowledge into account. It cannot explain the cause of children's misconceptions. Moreover, repair theory assumes that an impasse is a result of incomplete

knowledge - it does not take other causes into account, for example, incorrect application of complete knowledge.

Finally, the theory only accounts for a small proportion of observed errors. Brown and VanLehn acknowledge that due to constraints like those mentioned above, repair theory could predict only 23% of the known subtraction bugs. As pointed out by Evertsz (1991), due to this low coverage, it seems likely that there will always be some bugs that repair theory will not be able to account for.

2.2.5 PIXIE

PIXIE is an extension of a previous modelling system, called the Leeds Modelling System (LMS, Sleeman and Smith, 1981). The system is for the domain of algebra. PIXIE has a library of rules containing the bugs (called malrules) and the correct rules. The student model is made up of rules from this library. PIXIE is able to infer new malrules to account for student errors which were not encountered before (i.e. where there were no rules in the library corresponding to these errors).

This was done using the student's protocols, working backwards from the student's answer to the question. In the algebra domain, since there are several steps involved before the answer can be reached, and at each step, there are several alternatives that a student can take, there is a huge space to search. While searching the paths from the answer to the problem, PIXIE first uses its library of rules. If the path can be constructed with correct rules (in the correct order), then the student's answer is correct. If they can be constructed using malrules and correct rules, then the student's errors can be explained in terms of the existing rules. The challenge occurs when the search fails, i.e. the complete solution-problem path cannot be explained. In this case, PIXIE hypothesizes a new malrule

which explains the last step in arriving at the question. The following example shows how this is done:

A) X = 3 (student's answer)

B) X = (3-n) + n (n=8)

C) X = 8 - 5

D) X + 5 = 8

E) X + (5-n) + (n) = 8

(Instantiate n using lhs of the question)

F) X + 2 + 3 = 8

G) 2X + 3 = 8 (question)

At each step in inferring the student's path, one of the following alternatives is applied:

i) One of the existing rules is applied, backwards. For example, step D in the above example is derived using the (correct) rule, lhs +/- M = rhs ==> lhs = rhs -/+ M backwards (i.e. X + 5 = 8 <== X = 8 - 5).

ii) Focussing heuristics, selected from a set of ten, are applied in order to get the equation to have the same form as the 'target' equation (i.e. the question). For example, at step B, the following heuristic rule is applied:

IF lhs(eqn) is not equal to lhs(target)

AND rhs(eqn) is not equal to rhs(target)

AND rhs = i

THEN replace (i, (i-n) + (n))

iii) If neither of the above can be applied, then a new rule is inferred. In the step from F to G in the above example, the system makes the inference that the student went from step G to step F. Hence, AX + B = C ==> X + (A + B) = C. Ignoring the symbols which are not changed, the system infers that the malrule that the student applied was AX ==> X + A.

There are three problems with this approach:

i) It would only work if the student's error is in the first step of his/her solution (in the above example, AX + B = C ==> X + (A + B) = C), which obviously is not general enough - it is equally likely that the student would make an error at any other step in the solution.

ii) It assumes that malrules are independent of each other, and that any error is due to a single malrule. In addition, it assumes that the errors are logical - it finds a new malrule through a set of logical steps. If an error was caused by two malrules, then PIXIE will attempt to derive a single new malrule to explain the child's "buggy" behaviour. If it cannot, it concludes that the child is exhibiting inconsistent behaviour. Such malrules may be logically plausible, but not necessarily psychologically plausible.

iii) A large number of heuristics (like the one at step B above) need to be provided. This leads to a huge search space in order to select an applicable heuristic.

On the one hand, PIXIE does not require a library of malrules to be provided before a student's malrules can be identified. On the other hand, it still needs some kind of 'domain analysis' - that is, the set of heuristics.

A further limitation of LMS and hence of PIXIE is the categorizations of the rules and malrules as applicable to certain 'levels' only. Since there are such a large number of rules and possible malrules in algebra, there are a huge number of combinations of these rules from which to search a

student's model. To solve this combinatorial problem, Sleeman (1983) assumed that the domain of algebra skills is hierarchical and can be 'split' into independent subskills. He divided the total set of rules into smaller sets, each corresponding to a 'level' in the hierarchy. Modelling proceeds by first considering the subskills at level 1 and then level 2, etc. as the student presumably proceeds to more complex subskills. One can understand how this got round the combinatorial problem, but, as pointed out by Evertsz and Elsom-Cook (1990), this approach was later found to be flawed, because students who have mastered a given subskill on its own cannot always apply it when it is part of a more complex problem. Sleeman (1983) himself, in a description of one of the experiments with LMS, states "This experiment showed that this was not a valid assumption" (Sleeman, 1984, p. 389).

2.2.4 State Constraint Theory

Ohlsson and Rees (1988) propose a theory of conceptual understanding and its role in the learning and execution of arithmetic procedures. The theory is called the state constraint theory of understanding. The hypotheses of the theory are as follows:

- The type of declarative knowledge that is essential for procedural learning is knowledge of general principles. A principle is defined as abstract knowledge that consists of assertions that apply to every case. For example, 'subtraction decreases numerosity' applies to every instance of subtraction.

- Principles constrain the possible state of affairs.

- A cognitive performance is a heuristic search through a problem space.

- Procedural knowledge consists of collections of search heuristics.

- Learning begins when an incorrect or incomplete procedure generates a search state that violates one or more principles.

- A faulty procedural rule is revised on the basis of information in the learner's principled knowledge.

Ohlsson and Rees implemented a computer model of the theory, called Heuristic Searcher (HS). A procedure in HS consists of a collection of production rules. The condition of a rule is matched against the current search state. The action consists of a problem solving operator. Principles are encoded as state constraints which are criteria that a search state has to satisfy in order to be correct. A state constraint C is an ordered pair of patterns, represented as <Cr * * Cs>. Cr, called the relevance pattern, determines the class of search states to which the constraint is relevant. The right-hand pattern Cs, called the satisfaction pattern, encodes the criterion that a state must match in order to satisfy the constraint. Whenever a constraint is relevant, it has to be satisfied. A heuristic search mechanism, also represented as production rules, compares each search state with the constraints and decides whether they are satisfied. States that violate one or more constraints are inconsistent with the system's knowledge, and from this the system knows that its procedure is incorrect, and that it needs revisions. It is the constraint violation that triggers the system to learn. If a state violates some constraint, HS applies its learning mechanism to the rule that produced the constraint violation, and revises it, replacing it with more constrained rules. After revising a rule, HS returns to the initial state of the current problem and tries to solve it.

The authors describe the construction of a general counting procedure from the state constraint representation of the principles of counting. An incomplete set of rules represent the procedural knowledge for counting. The set of rules generate incorrect behaviour, since they are incomplete. Principled knowledge is represented by a complete set of state constraints.

The complete representation of principled knowledge and the incomplete representation of procedural knowledge implies that an unsuccessful attempt at counting is not due to a lack of any principled knowledge, but is a result of not being able to apply that knowledge correctly. Hence, a task is unsuccessful because the procedure is incomplete, which leads to constraint violations. The learning mechanism completes the procedural knowledge by adding the principled knowledge that was violated to the rule that caused the violation.

For example, the following rule is one of six rules in HS that represents standard counting:

((Number N1) (Current N1)) ===> (Assert N1)

where (Assert N1) asserts that the number N1 is the answer. The task of standard counting can be described as follows. To count a set of unordered objects is to repeatedly select an object from that set, increment the current number, and associate the new number with the selected object. When all objects in the set have been associated with numbers, the last number to be associated with an object is asserted to be the answer to the counting problem. The above rule will assert that the current number is the answer, even when there are still objects left to be counted. When this happens, the constraint, (Answer N1) * * (Associate X1 N1), is violated, since the relevance criterion, (Answer N1) is satisfied, and the satisfaction criterion, (Associate X1 N1), which associates the number N1 with the object X1, is not. The satisfaction criterion is added to the initial rule and HS returns to the initial state and tries to do the counting task again.

The revised rule violates the constraint, (Answer N1) * * (Not (Member X2 ToCountSet)) (Not (Associate X2 N2)), which states that a number is the answer to a counting problem only if there are no objects which are members of the to-be-counted set and which have not been associated with some number. The rule does not contain the knowledge that it has to wait

for all the objects to be counted. It prematurely asserts that the current number is the answer as soon as that number has been assigned to an object. Hence in a set of 10 objects to be counted, the system would assert 1 as the answer as soon as 1 has been assigned to the first object. Ohlsson and Rees claim that during learning, the model commits the types of counting errors that are observed in children's performance. However, it is hard to believe that children make errors like the one just described - the answer for a set of 10 objects to be counted is 1. Hence, Ohlsson and Rees' claim that the model commits the type of errors that children make can be questioned. As a result of the above constraint violation, the learning mechanism adds the satisfaction criterion of the constraint that was violated as a condition to the rule. Thus, the final rule, with the learned conditions is:

((Number N1) (Current N1) (Associate X1 N1) (Not (Member X2 ToCountSet)) (Not (Associate X2 N2))) ===> (Assert N1)

that is, assert a number as an answer only if there does not exist an object which is a member of the to-be-counted set and which has not been assigned a number.

One of the strengths of HS is that it uses principled knowledge. Given a set of rules representing incomplete procedural knowledge for performing a task, and a set of constraints representing the complete principled knowledge, HS learns to perform the task by completing its procedural knowledge. In addition, it is capable of explaining the cause of certain types of errors - those that are caused by constraint (principle) violations. However, it is not able to explain other types of errors, for example, errors that might occur in applying or utilizing a procedure. It will also not be able to account for those errors that are caused by children's misconceptions, since it does not have any knowledge of misconceptions. Including misconceptions would require substantial changes to the state constraint theory.

A further limitation of HS is that it includes only one type of learning, that is procedural learning based on principled knowledge. Any learning that might be triggered as a result of procedures is totally ignored. Furthermore, it is assumed that a complete set of principled knowledge is known before carrying out a procedure that might be dependent on the principles. It is also assumed that this set of principles can be analysed and clearly distinguished from procedural knowledge. These are huge assumptions. The review in section 2.1 above shows evidence of views that are in conflict with these assumptions.

2.2.6 Automated Cognitive Modelling

The Automated Cognitive Modeller (ACM; Langley, Ohlsson and Sage, 1984) applies the machine learning technique of learning from examples to generate student models. The modeller was developed as a general tool for cognitive modelling, although the authors describe it mainly in the context of the domain of subtraction. The output from the model is a description of student behaviour like that produced by LMS.

ACM requires two basic inputs:

i) a problem space, and

ii) some information about the behaviour of the student to be modelled. This consists of a set of problem-final answer pairs.

The problem space consists of:

i) the representation of individual states (e.g. for the domain of subtraction, the initial state is the problem, the final state is the solution to the problem),

ii) a set of operators for moving from one state to another (e.g. add-ten, shift-column, decrement, etc.) and

iii) a set of rules that state the conditions under which the operators may be applied. For example, the following is a condition-action rule for the operator add-ten

add-ten

If you are processing column1,

and number1 is in column1 and row1,

and row1 is above row2,

then add ten to number1.

Given a student's answers to a set of problems and the initial set of condition-action rules, ACM needs to determine the sequences of operators that led to the student's answers and infer the conditions that will reproduce these sequences on the same problems. In other words, the student modelling task is reduced to the problem of determining whether a given operator was used by the student, and if so, determining the conditions under which it was used.

Using the initial set of general rules (provided as input iii) above), and the student's answer to a problem, ACM generates a search tree consisting of the observed (student's) answer along with many others. Figure 2-1 shows the search tree for determining a student's path for the problem 93 - 25 = 72 (This student has the 'smaller from larger' bug). Each step along the path leading to the student's answer is labelled '+' (a positive instance of the operator). Those steps leading one step off the solution are labelled '-' (negative instance of the operator). This process of searching the solution paths is repeated for the other problems. The solution paths provide a set of instances of operator applications. For example, examining the search tree

in Figure 2-1, there are two positive instances (5 - 3 and 9 - 2) and two negative instances (3 - 5 and 2 - 9) for the find-difference operator, one positive instance for the shift-column operator, two negative instances for add-ten operator and one negative instance for the shift-left operator.

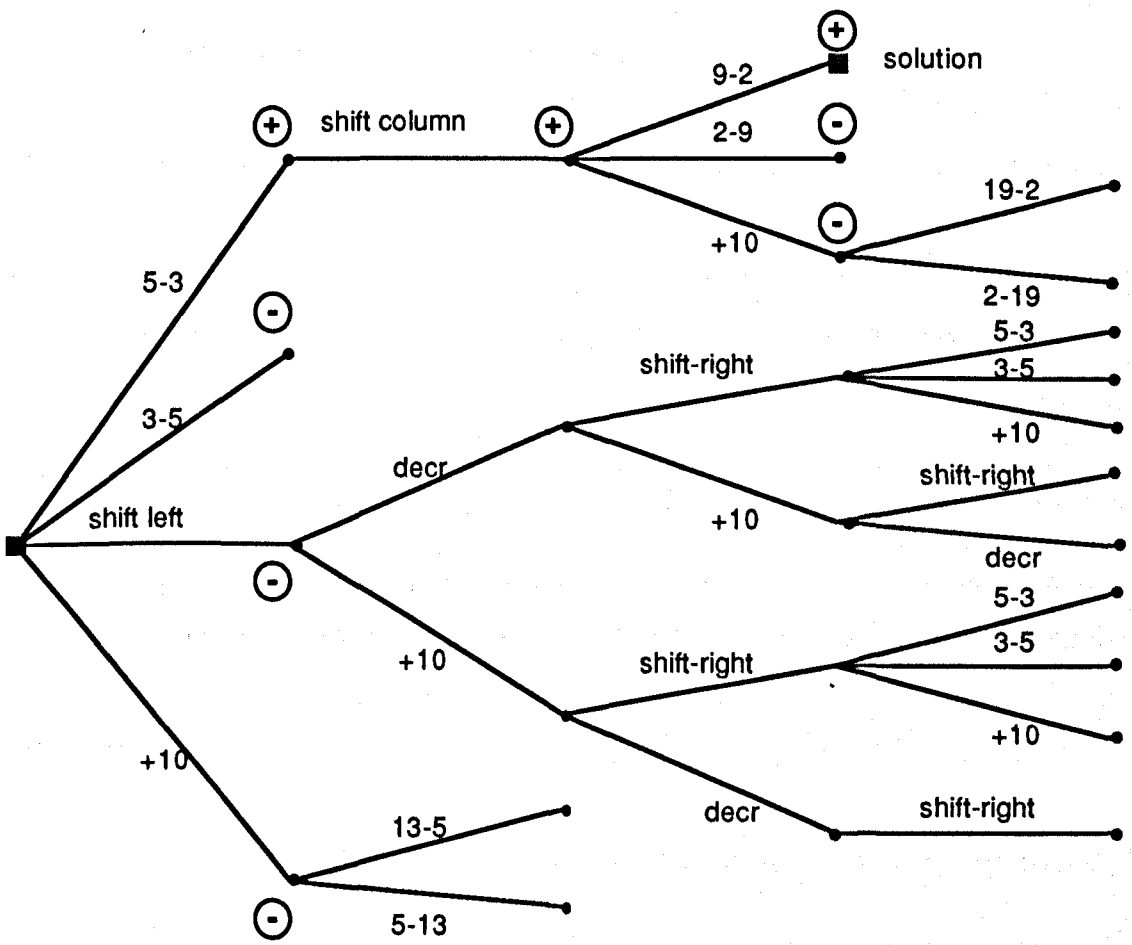


Figure 2-1. ACM's search tree for the problem 93 - 25 = 72 (Wenger, p. 214)

Having obtained a cumulative set of positive and negative instances for each operator from the search trees of the set of problems, the conditions under which the operator is applied are determined using a discrimination learning method.

For determining the conditions, a set of tests have to be provided (e.g. greater N1 N2, above R1 R2). The tests are used for discriminating positive and negative instances of an operator. Those tests, that satisfy only positive instances of an operator, make up the conditions of the operator applications (such conditions-operator rules make up the production-system model of the student's behaviour).

ACM's modelling capability depends on the initial domain knowledge provided, which in turn depends on some knowledge of malrules in the domain. This is because the operators have to account for all possible actions that students may take. Another limitation of ACM's modelling technique is that it would work only for structured domains, or for domains in which the intermediate steps that a student can take are known. In addition, due to the huge search space, the approach would be practical only for those domains which have a small number of ways of arriving at the solution (i.e. a small number of operators). ACM's learning algorithm is not incremental, that is, it needs a *set* of problem solutions, and hence is not suited for tutoring.

Both PIXIE and ACM are diagnostic systems, not implemented for tutoring. Neither of them say anything about how the student models will be used for tutoring. If one wishes to implement a tutoring system using these approaches to modelling, one has to go through two phases:

i) the diagnostic phase

ii) the tutoring phase plus the diagnostic - this would be implemented using the student models obtained in the above phase.

The second phase still includes the diagnostic phase in order to discover any new malrules that had not been encountered in phase i). The appropriate remediation corresponding to such new malrules would then have to be programmed in the tutoring system. In addition, neither of the

modelling techniques attempt to model the learning process of the student. Hence, they cannot explain how the malrules were first acquired. Finally, both the approaches are general enough to be applied only to domains which are structured and those which have a small number of possible ways of problem solving.

2.2.7 Summary

The models in PIXIE and ACM do not aim to model human learning processes. Their main purpose is to automate the construction of student models.

Repair theory and HPM model procedural learning. They both fail to take conceptual knowledge into account. To my knowledge, Greeno, Riley and Gelman's planning net representation and Ohlsson and Rees' application of constraints are the only models of arithmetic learning which represent conceptual knowledge and its links with procedural knowledge. In both cases, the connection between these two types of knowledge is made by assuming that performance is a consequence of conceptual competence. The literature on empirical investigations of arithmetic knowledge reveals that other assumptions, for example that conceptual knowledge is a consequence of procedural knowledge, are equally valid. This leaves the explanation of the learning of concepts from procedures for further work.

Ohlsson and Rees' state constraint theory focusses on procedural learning based on principled knowledge. Their computer model learns procedures assuming that a complete set of principled knowledge is known. A more complete model would be one which integrates a model that explains learning of the appropriate principled knowledge as well. Of the systems described above, Greeno, Riley and Gelman's action schemata theory is the only possibility for such an integration, since neither Repair theory, HPM, ACM nor PIXIE use principled knowledge or explain the acquisition of

procedural knowledge. However, the action schemata theory has not been implemented. Hence, computational models of conceptual learning remain for further work.

2.3 Conclusions

More complete and general models of learning are needed in order to explain children's learning. The computational models reviewed above each focus on some subset of the issue of learning. For example, Ohlsson and Rees model only one, very specific, type of learning. With further research on models of learning, and their applications to different tasks, the present models of cognition can be improved.

None of the above computational models of learning cover the application of what is known to new situations. For example, the transition from counting (1 set of objects) to addition (counting 2 sets). The research presented in this thesis explores this particular type of learning and applies it to the transition from the ability to solve 2-term addition problems (e.g. 4 + 5) to that of 3-term problems (e.g. 4 + 5 + 2). Before being able to model a mechanism of the learning process, some 'static' models were needed. These have been constructed using production rules, and are based on empirical work. The following chapter describes empirical studies that were carried out to investigate children's performance on tasks related to the concepts of commutativity and associativity.

Chapter 3

STUDIES OF COMMUTATIVITY AND ASSOCIATIVITY

3.1 Introduction

The development of many arithmetical skills depends on an understanding of basic principles like commutativity, associativity and distributivity[1]. However, little research exists on the acquisition of these concepts, their applications and how they are related to other skills in arithmetic.

This chapter is an account of three studies[2] that were carried out to investigate the development of the concepts of commutativity and associativity in children aged four to twelve years. The three studies consisted of a pilot, a main study and a longitudinal study. The aims of the studies were:

i) To identify the stages that children go through in acquiring the concept of commutativity for addition of integers.

ii) To find out if commutativity can be taught.

iii) To achieve some understanding of why children generalize commutativity to subtraction.

---

[1] Distributivity: $a(b + c) = ab + ac$; $a(b - c) = ab - ac$; $a(b * c) = ab * ac$ for all real values of a, b and c.

[2] Note that these were exploratory exercises aimed at investigating surface level behaviour of children, as opposed to standard psychological experiments.

iv) To gain insight into children's progression from commutativity to associativity. Logically, commutativity is a prerequisite of associativity. However, children might not progress in this way. There might be an overlap in the acquisition of the two concepts (i.e. stages where both are being learnt at the same time, as opposed to the requirement of a complete understanding of commutativity before going on to the early stages of acquiring associativity).

v) To study the connections between children's conceptual knowledge and their problem-solving strategies.

vi) To study the development of strategies over time.

In the next section of this chapter, the pilot study is reported. Section 3.3 is an account of the main study, and the longitudinal study is described in section 3.4.

The term *grouping* will be used to mean a combination of commutativity and associativity. This is so that strategies like 3 + 4 + 7 = 3 + 7 + 4 and 3 + 4 + 7 = 7 + 4 + 3, i.e. those which do not necessarily operate on the numbers from left to right (but in any order), can all be described by one term.

3.2 A Pilot Study

One of the aims of the study was to establish the age range in which children's concept of commutativity develops. The study examined the use of commutativity by children between the ages of six and ten. Two sets of tasks were administered to find out whether children knew the concept of commutativity or not.

3.2.1 Method

Subjects

22 children, aged between six and ten, from a state primary school in Milton Keynes were interviewed and observed while carrying out some arithmetic tasks involving commutativity. The sample consisted of 9 girls and 13 boys. 7 of them were in the age range six to seven, 8 in the range seven to eight, 3 in the range eight to nine and 4 in the range nine to ten years. The mathematical abilities of the subjects were decided by the teacher and noted as either high, medium or low.

Task 1

The first task was derived from a study in the form of a "Quick Look" game carried out by Baroody and Gannon (1984). In this task, pairs of addition problems were written on cards (3" by 5"). The second of the pair of problems was written next to the first one. The pairs were of one of the following three types:

i) 10 commutative pairs (e.g. 3 + 4 and 4 + 3),

ii) 2 identical pairs (e.g. 3 + 4 and 3 + 4) and

iii) 10 pairs of problems with different sums (e.g. 3 + 4 and 5 + 1).

The identical pairs were included for deciding the success of any ambiguous students, i.e. those students who might say "the same" only for such problems. These problems provided obvious cases of being the same. The problems of the third type provided obvious cases of being different. Appendix 1 lists the problems in the order they were presented.

The children were given the following instructions, and practice on a couple of problems to put emphasis on the *sums* of the problems: "You won't have enough time to work out the answers - you'll just have a quick look and tell me if the adding problems would give the same answer. Now let's try some problems, shall we? 2 + 5 and 3 + 6 - Do you think they will add up to the same or different answers?". They were then shown each card one at a time, for a short time - for a quick look only, and were asked whether the two problems would give the same answer or different.

A child was scored as successful if s/he was correct on 9 or 10 of the 10 commutative problems and if s/he was correct on 19 or more of the 22 problems overall (to make sure that they were correct on different-total and identical problems as well). Otherwise, s/he was scored as unsuccessful.

Task 2

This task involved testing different ways of finding out children's levels of understanding of the concept of commutativity. Children were given several addition problems (most of them being commutative pairs, i.e. x + y followed by y + x). To begin with, all individuals were given 3 + 4. Depending on their performance on this problem (and on other problems), the numbers in the following problems were chosen. They were not all given the same set of problems for several reasons. Firstly, since the subjects were from a wide range of abilities (different ages), some problems that were too easy for some children were difficult for others and vice-versa. It was found that some children gave answers to simple problems immediately (memory recall). More difficult problems had to be given to them to find out more about the way they computed the answers. Secondly, some children exhibited the behaviour that the experimenter was interested in, in fewer problems than others. Thirdly, some children needed more practice with addition problems than others.

The problems were written on paper (using abstract symbols like 4 + 5), one at a time. They were read out with "plus" and repeated using "add" and "and" if needed. The experimenter used the individual children's terminology whenever it was revealed. The children were told that they could use whatever they liked, and could solve the problems whatever way they liked. Some children had to be given concrete or real world analogies to the abstract problems. For example "4 plus 3. Say you had 4 apples and I gave you another 3. How many apples will you have?", and shown this with counters (unifix cubes). The strategies used by the children were noted. Questions like "How did you get that?" helped to reveal their strategies.

To find out about their knowledge of commutativity, they were interrogated as follows:

a) "Which is more: 3 + 4 or 4 + 3?", "Why?".

b) After they had written down the answer for the first of a commutative pair of problems, they were asked "Now, can you tell me if this will add up to x (where x is the child's answer) - the same as or different ...", "Why do you think it will be the same (or different, depending on their response)?". This subtask is similar to task 2 conducted by Baroody and Gannon (1984). In addition, a pair of wrongly-answered, large-numbered problems (like 1023 + 4970 = 5985 for older children, and 130 + 485 = 550 for the younger ones) were written down, read and then the children were asked "what do you think 4970 + 1023 (or 485 + 130) will be?". Furthermore, it was noted whether the subjects computed the sums for the second of the commutative pairs of problems, or whether they copied the answers from the previous problems.

c) Those who showed knowledge of the concept were tested to see if they generalized it to all numbers: "If I swapped the two numbers around, will the answer always be the same?", "Even for very large numbers?", "Do you know why that is?".

d) Children who showed signs of understanding the concept were given a pair of subtraction problems to find out if they generalized it to this operation.

The subjects were classed as successful, marginally successful or unsuccessful. At least two of the following criteria needed to be satisfied for a child to be judged successful:

i) The response to question a) above included "same" or its equivalent (e.g. "they are both more").

ii) The response to question b) was "same" or the same answer as that in the previous problem.

iii) The child generalized the concept of commutativity to all numbers.

iv) S/he generalized the concept to subtraction.

v) S/he consistently used the answer from the previous problem for answering the second of a pair of commutative problems.

vi) His/her explanations included statements which referred to the *sums* of two problems being the same, e.g. "They are the same, but you have swapped them around and it equals the same number".

A child was classed as marginally successful if at least two of the following criteria were satisfied:

i) S/he did not think the answers to the two problems in question b) above were the same or if s/he gave the correct answers, but with hesitation.

ii) S/he was not sure of his/her answer to question c).

iii) S/he applied the concept to small numbers only or to concrete examples only.

iv) His/her explanations included statements that referred to the two addends and not to the sum; for example "because 2 was there and 7 was there".

v) His/her performance showed no evidence of copying the answer from the previous problem or s/he copied sometimes and worked it out sometimes.

A child was classified as unsuccessful if one of the following conditions was satisfied:

i) The response to question b) was "different" or any other answer apart from the answer to the previous problem, or if s/he counted or started working out the answer.

ii) S/he was not classed as "successful" or "marginal".

Table 3-1 below shows the evaluation criteria for overall success based on the scores in tasks 1 and 2.

Table 3-1. Evaluation criteria for overall success

Task 1	Task 2	Overall success
success	success	success
success	marginal	marginal
success	unsuccess	unsuccess
unsuccess	success	marginal
unsuccess	marginal	marginal
unsuccess	unsuccess	unsuccess

3.2.2 Results and Discussion

For the question "which is more, 4 + 3 or 3 + 4?", responses other than "the same" or its equivalent were ignored in the analysis. This was because it was found that some children were confused. They said that one of the pair of problems was more than the other because they thought they had to pick out one of the two. Perhaps "Is one of these more, 4 + 3 or 3 + 4?" would have been a better question. Some of the children who thought they had to pick out one of the problems as "more", chose the problem with the larger addend first and gave explanations like

"because 4 is more"

"because that's got 4 + 3 and 4 is before 3".

Hence, for such children, this question did not provide any certain information. However, it did reveal some of those children who definitely knew the concept; for example, KF (Table 3-2) exclaimed:

"They are the same. Its just the other way around".

Twelve out of the 22 subjects were successful on both the tasks, seven were marginally successful overall and three were unsuccessful (as shown in Table 3-2). In Task 1, despite the experimenter's emphasis on the sums/results of the problems, there were some children who might have responded to the similarity of the addends and not to the sum. This problem was encountered by Baroody and Gannon (1984). An example is JV (Table 3-2) who was completely unsuccessful on Task 2. When asked why she had said 4 + 5 was the same as 5 + 4, she replied:

> "Because that and that (pointing to the two 5s) are the same and that and that (pointing to the 4s) ...".

JV's response might be interpreted as ambiguous. It could mean that the problems are the same because they have the same addends, or that the sums are the same because the problems have the same addends. However, the latter is unlikely because of her performance on Task 2.

SE (Table 3-2), who did not have any knowledge of commutativity, referred to something other than the sum of the addends:

> "They are different because they are the other way around" (SE's protocol can be seen in Appendix 2).

These examples show that Task 1 alone would not be a good measure for assessing children's knowledge of commutativity. The data (Table 3-2) indicates that Task 1 produced a higher success rating than Task 2 about 1/3 of the time (Task 1 overestimated performance relative to Task 2 seven out of twenty-two times) unlike Baroody and Gannon (1984) who found that Task 1 did not systematically underestimate success relative to their Task 2 (Task 2b of the present study).

Table 3-2. Summary of performance on commutativity

SUBJ	AGE	T1	T2	OV	OBSERVATIONS
KF	9.1	S	M	M	COL; used comm. for small numbers only; "because its the same as that, but its the other way round"
NC	9.0	S	S	S	COL; "its the same, but the other way around"
MC	9.3	S	S	S	COL; "they are the same but you have swapped them around and it equals the same number"
KT	9.3	S	S	S	COL when not using blocks for counting; CAF when using blocks (for problems with sums > 10)
MT	8.0	S	S	S	COL
DM	8.1	S	S	S	-
T W	8.0	S	M	M	CAF
ML	7.6	S	S	S	COL; "they are just the same but opposite ways around"
StD	7.0	S	S	S	COL; "because I remember it was the same , the numbers were just the other way around"
W A	7.0	S	M	M	COL; calculates again for larger numbers
DS	7.0	S	S	S	COL; uses blocks; "its the same as that one"
FJ	7.0	S	M	M	CAF
DA	7.5	S	S	S	COL
ZD	7.0	S	S	S	COL; "just copied because its the same problem but in a different way"
DC	7.0	U	M	M	COL; did not use comm. for the first 2 pairs of problems, but did for the rest
DG	6.9	S	S	S	COL; "because they are the same numbers"
SD	6.0	S	S	S	COF
PM	6.0	S	M	M	COL; uses comm. for small numbers only

GL	6.0	S	M	M	CAF; could not add without blocks
JV	6.5	S	U	U	COL; worked out the answers for each problem in Task 2
SE	6.0	U	U	U	CAL sometimes and CAF sometimes; did not use comm. at all
CF	6.0	U	U	U	CAF; used blocks

OV stands for overall performance.
S, U and M refer to success, unsuccess and marginal success respectively.
CAF - count all from the first addend; CAL - count all from the larger addend;
COF - count on from the first addend; COL - count on from the larger addend.

The results show that some children apply the concept to small numbers only. This is probably due to the fact that they are experienced with using fingers (numbers up to 10) and concrete objects which normally represents small numbers. The analogy of fingers, is often revealed by children, even when they are not using them overtly. One child, for example, revealed "I imagined my fingers in my mind".

One child, DC did not succeed on Task 1, but showed marginal success on Task 2. On Task 2, he did not apply commutativity to the first two pairs of problems (recomputed the sums to the second of the pairs of problems), but did on the third pair: "copied it off there because 9 was there and 7 was there". It is possible that DC learnt the concept during the second task. This finding is consistent with Baroody and Gannon's (1984) results.

SE (Appendix 2), sometimes disregarded addend order and started from the larger addend. This is consistent with previous findings (e.g. Carpenter and Moser, 1983). It is interesting that SE did not think that 6 + 4 was the same as 4 + 6, and yet sometimes ignored addend order. JV provides further evidence of using the 'count on from the larger addend' (COL) strategy without a complete understanding of commutativity. This is consistent with Baroody and Gannon's (1984) finding of children who used or discovered COL or 'count all from the larger addend' (CAL) strategy but did not appreciate commutativity.

On the other hand, there were children who understood commutativity but did not apply it (or only applied it occasionally). Examples of such subjects are KT, SD, TW, FJ and GL. Fuson, Secada and Hall (1983) also found this in an analysis of the transition from counting all to counting on. From their study, they found 9 out of 45 children who used count all, and demonstrated all the subskills required for the transition from count all to count on, but did not use count on. The issue of procedure first versus conceptual knowledge before procedure is further discussed in section 3.3.2.

The results in Table 3-2 show that 68% of the children used the COL strategy. This is because most of the subjects were quite old and hence more experienced at such problem solving.

The results also show that children who were not successful were less than seven years old. This revealed the need to study the development of the concept with children younger than seven. This study also pointed out that there are some children as old as nine who are not completely successful (e.g. they apply the concept to small numbers only). This led to the inclusion of children up to the age of twelve in the second study.

3.2.3 Conclusions

From Task 1, it was found that students' answers "same" or "different" could be interpreted ambiguously. In addition, the tests in the second task were enough to judge a child's level of success. In fact, as can be seen from Table 3-1, the overall score is the same as the score on Task 2, except for one case, where unsuccess on Task 1 and success on Task 2 leads to an overall marginal success. Besides, this exceptional case was not encountered (Table 3-2). Hence Task 1 was abandoned in the next study.

The study showed that once children knew the COL strategy, they preferred it to the other strategies. It also showed a trend in the development of strategies and of the concept of commutativity with age.

3.3 The main study

A more comprehensive study into the development of the concept of commutativity was carried out. This was done in a rural school in Fiji (note that the location is conceptually irrelevant in the study).

The aims of the study were to:

i) gain insight into the stages that children go through in the acquisition of the concept of commutativity,

ii) find out if the concept can be taught,

iii) investigate why some children overgeneralize the concept to subtraction and

iv) examine the extension of knowledge of the concept to 3-addend problems.

3.3.1 Method

Subjects

The subjects comprised 105 children (49 boys and 56 girls) between the ages of five and twelve. They were from 14 classes. The classes are divided according to the children's ages; for example, children between six and seven years old are in class 1, between seven and eight-years-old are in class 2, between eight and nine are in class 3, etc. There are two classes in

each age range. The five-to-six-year-olds are kindergarten students. The sample consisted of 4 five-to-six-year-olds (originally, 15 five-to-six-year-olds were interviewed, but it was found that they were not competent enough for the study - most of them could not add; some could not even count up to 10), 31 six-to-seven-year-olds, 33 seven-to-eight-year-olds, 22 eight-to-nine-year-olds, 8 nine-to-ten-year-olds, 5 ten-to-eleven-year-olds and 2 eleven-to-twelve-year-olds.

The kind of activities carried out in the classrooms in Fiji during mathematics lessons are similar to those in England; the contents of mathematics textbooks are similar; there is no evidence of student-teacher relationships in the two schools being significantly different. The amount of concern from parents towards their children's mathematics education might be slightly higher in the English school ("my mum taught me"), whereas in this particular rural school in Fiji, parents tend to leave mathematics education for the teachers at school. A major difference in the two schools is the age at which children start formal schooling (hence the transition from informal experiences to the formal system). For the Fijian school, the earliest age is five years, when they start kindergarten. There were some children in class 1 who did not even attend kindergarten. In England, much younger children attend nursery or kindergarten. As a result of this, children from the Fijian school might achieve a certain knowledge/performance level at a slightly higher age than children from the English school. A second difference between mathematics education in this rural school in Fiji and that in England is the medium of communication. The language used in the study school in Fiji is Hindi, which is the mother tongue of all the subjects. At the time of the study, class 1 children were involved in learning to count and writing down numbers; class 2 children were practising addition; class 3 was being taught subtraction concepts, place value and partitioning (formal algorithm); classes 4 and 5 were onto multiplication and division and class 6 students were on more advanced topics like area.

Materials

The children were given a set of pairs of addition problems like 4 + 5 and 5 + 4. The numbers in the problems were randomly selected. Appendix 3 contains a list of a typical set of problems. The children were not all given the same set of problems. The problems were written on paper, one at a time, and read out. The difficulty level (the size of the numbers) of the problems depended on their performance on the previous problems. If a child found the problems difficult, then the sizes of the addends were decreased. If the child applied commutativity, then larger addend problems were given in order to find out whether s/he applied the concept to larger numbers. After doing the set of addition problems, a pair of subtraction problems were given, where the order of the two numbers in the second problem were reversed.

The 3-term problems fell in one of the following six categories:

i) addition only (e.g. 5 + 8 + 5),

ii) subtraction only (e.g. 16 - 10 - 5),

iii) multiplication only (e.g. 7 * 2 * 4),

iv) division only (e.g. 12 / 6 / 2),

v) combination of addition and subtraction (e.g. 3 + 5 - 3) and

vi) combination of multiplication and division (e.g. 5 * 3 /3).

These problems are normally written with parentheses, for example, (8 + 5) + 5; The problems were written without the brackets because the aim of the study was to see if the students used grouping. To avoid the representation of the problems being another variable, the children were given a word

problem first (e.g. "suppose you bought apples worth 20c, lollies worth 15c and a packet of crisps worth 14c. How would you write a sum to find out how much you needed to give to the shopkeeper?") to find out how they represented such problems. None of the subjects used brackets, hence their representation was consistent with mine.

The children used fingers for counting. Some of them used rulers. Some icelolly sticks were also provided. A tape recorder was used to record the interviews.

Procedure

Children were given arithmetic problems (of the type in Appendix 3) and were observed solving them. They were interviewed, as in Task 2 of Study 1 (with the deletion of the question "which is more?"), to get details of the skills and strategies they were employing. The work was carried out in two stages:

i) a study of commutativity and

ii) a study of transfer of knowledge of commutativity to solving 3-addend problems.

To identify the different levels of understanding of the concept of commutativity, children at different stages/levels (that is, some in kindergarten, some in class 1, some in class 2, etc.) were studied. In addition to the commutative pairs of problems, 63 of the children were also given a pair of subtraction problems like 6 - 5 and 5 - 6, in order to find out if they would apply commutativity to these as well, and if they did, then why. The subjects who were definitely identified as unsuccessful on the commutativity problems were not given the subtraction problems (since if they did not have any knowledge of commutativity, there is no question of generalization to subtraction).

Seventy-seven of the students who were studied for the stages of development of commutativity, were also studied for their performance on 3-addend problems. The sample size of seventy-seven students was a result of ignoring those students who were not competent enough for the study (for example, those who did not know what to do or how to proceed on the 3-addend problems. Some of these students had explicitly stated that they did not know how to do these problems). Eleven of the older children from the sample of seventy-seven, who had knowledge of multiplication and division were given problems in categories ii, iii, iv, v, vi (see the section on materials above) as well. This procedure was carried out to establish if there is a relationship between children's performance on 2-addend problems and that on 3-addend problems.

3.3.2 Results and discussion

Performance levels of commutativity

The study revealed several levels of performance of the concept of commutativity. There are the basic levels where children use the COL strategy for addition, because it is a faster means of arriving at the answer, without showing any evidence of conceptual knowledge of commutativity. There are the fully-developed stages where knowledge of the concept is applied to more complex situations (e.g. application to 3-addend problems and invention of informal procedures for solving problems). The following levels in the development of commutativity (of addition of integers) are proposed as a result of the performances of the subjects in the study:

i) **Order-irrelevance principle** (Gelman, 1977) - while assigning tags to objects in a set, it does not matter which tag is assigned to which object.

ii) **Implicit knowledge** - a level at which children might possess knowledge of the concept, but it is implicit (they cannot articulate it - some do not have the language/vocabulary to describe it, and yet some of them are able to apply it). For some children, their knowledge might be at a level which does not allow them to make use of it completely. Hence, children at this level compute the answer to x + y after having done y + x, instead of copying the previous answer.

iii) **Commutativity** - the realization/explicitness that a + b is equivalent to b + a (for all values of a and b), and the use of the concept. This may proceed in steps (not necessarily in this order):

a) concrete examples only,

b) small numbers only,

c) abstraction and

d) generalization to all numbers.

The progression here is quite complex. The four steps can all be interrelated, and hence, a child can be at one or more of them at any one time. There are several possibilities of progression:

concrete small --> concrete large

concrete small --> abstract small

abstract small --> abstract large

concrete large --> abstract large

iv) **The extension of the application of commutativity.** At this level, children extended their application of the concept, for example to 3-term problems. There are several levels of application:

a) to concrete examples only,

b) to small numbers only,

c) generalizing to large numbers and to abstract examples,

d) inventing informal procedures based on the concept for solving more complex problems and

e) 3-addend problems (use of grouping).

At this level, like level iii), the different sub-levels of application may be interrelated.

Note that the above levels do not necessarily have a psychological or developmental status. They are just descriptions of performance. A child need not necessarily go from performance level i) to iv) in that order, nor does s/he necessarily go through each one of them. An attempt was made at categorising the subjects in the study into one of the four performance levels. This was done using the tape recorded protocols, the written problems and their answers and notes on the observations made during the interviews. The result for each subject is listed in Appendix 4. The details of the observations and measures used for deciding the levels are described below:

i) While counting out a set, children often recounted (for various reasons: e.g. to make sure or because they made a mistake or they lost track of their count or because the experimenter asked them to do so). When counting again, they did not necessarily assign the counting sticks the same tags as they did the previous time. One child, for example, picked up the sticks and put them in her hand as she counted them "1, 2, 3, 4, 5". When she recounted (she put them on the desk this time), she assigned the tag "1" to the stick to which she had assigned "5" previously. Although some children's behaviour revealed knowledge and use of this principle, this observation was not necessarily the criterion that was used in deciding a level i) child. Instead, a subject who was not categorised into the other 3

levels, was automatically classed as having performance level i). (Note that this level was identified more as a logical prerequisite for commutativity than as a result of performance behaviour). Note that the assumptions here are that all the children who didn't get to levels ii) and above did have the order-irrelevance principle and that this assumption is not important for the analysis in the rest of the thesis.

ii) This was the most difficult stage to identify. This level could have been split further into implicit or explicit knowledge and its applicability. This was not done because of the fuzziness of children's performance. In the case of a child who was put in this category, there was not a clear cut distinction between whether s/he had implicit knowledge and could use it or s/he had explicit knowledge but could not use it. The performance (use of the concept) on the commutativity problems only revealed that the subject had not reached stage iii) (the difficulty was to identify whether s/he had knowledge of the concept or not). Children who might have had the concept, but could not put it into words were categorised into this level. Note that no distinction has been made between commutativity and Baroody and Gannon's (1984) concept of protocommutativity (the order in which addends are dealt with does not make a difference in terms of the correctness of the sum). Hence, children who might have had knowledge of protocommutativity only, have been included in this category. Children who only have a concept of protocommutativity, do not realize that the result of 3 + 5 is the same as that of 5 + 3. They know that they get the answer in either case - whether they start from the first addend or from the second. This behaviour was noticed in children's performance on concrete tasks as well as on abstract examples. On concrete tasks, children at this stage, were observed not paying particular attention to the order of the addends, hence counting out either of the addends first. On abstract problems, there were several children whose level of understanding of commutativity was at this stage and were using the COL strategy while adding. When asked why they had started from the larger addend, and not from the first addend, their

replies did not reveal knowledge of commutativity: for example, "because it is easier", "because it is faster", "because x is larger than y". When subjects at level ii) were asked if the sums/results of a pair of commutative problems would be the same or not, they did not immediately say "same" (some started computing the answers). Even if they said "same", their explanations referred to the two addends, which did not necessarily extend to their sums. In addition, children at this stage succeeded on the large-numbered, incorrectly-answered problem (e.g. "If I told you that 130 + 485 = 550; now if I *swap* these two numbers around, 485 + 130, can you tell me what the answer will be?").

iii) It was common to find children at stage a) and at stage b) of this level. Due to the interrelationships between a), b), c) and d), it is difficult (if not impossible) to be precise about which one of these stages a child is at. Hence, a child was classified as being at level iii) if s/he was analysed as being at any one of these stages. A child was classified as being at this level if at least two of the following criteria were satisfied:

i) For the large-numbered, incorrectly-answered problem, s/he responded with the same answer.

ii) S/he generalized to subtraction.

iii) S/he used the answer from the previous problem for answering the second of a pair of commutative problems.

iv) His/her explanations included statements which referred to the *sums* of two problems being the same, e.g. "They are the same, but you have swapped them around and it equals the same number".

iv) A child was classed as being at level iv if s/he used grouping (did not necessarily carry out the sum from left to right) for solving the 3-addend problems.

Generalization

A total of 63 students' (see Appendix 4 for subject details) performance on pairs of subtraction problems, like 4 - 2 and 2 - 4 (after being tested on commutativity of addition), was examined (The results of the individual children's answers are listed in Appendix 4). Their responses fell into one of the following categories:

i) "cannot be done" or "not possible",

ii) "zero",

iii) applied commutativity to the second of the pair of problems (generalization). This was revealed by their responses, e.g. "it's the same" and

iv) subtracted the smaller number from the larger one. Children in this category gave explanations like "you take 2 from 4".

The number of students' responses in each category is listed in Table 3-3.

Table 3-3. Numbers of students' responses to problems like 2 - 4

not possible	0	generalize	smaller from larger
24	20	15	4

Note that this data is not based on a large set of problems, and so caution is needed on relying too heavily on this data. Having made this point, the

results shown in Table 3-3 are quite surprising. A higher proportion of students were expected to generalize. Eight of the students in the categories "0", "generalization" and "smaller from the larger" originally thought that it "cannot be done", but thought that this was like giving up on a problem, and hence attempted to arrive at solutions.

One of the reasons for children generalizing could be that they have been led to do so. They think that because the experimenter has given the example alongside the commutativity examples (i.e. her aim is to teach commutativity), this must be an example of commutativity as well. In addition, generalization is a way of learning. With the positive examples of commutativity that they have seen so far, there is no reason for them not to believe that everything is commutative. From this, one can say that students' interpretation of the situation, that is, how they think about it (Donaldson, 1978), and the type of examples are two possible variables responsible for their performance.

Another possible reason for generalization is that the children do not have a concept of negative numbers, which is the prerequisite knowledge for solving problems like 4 - 7, so they attempt a repair. One of these repairs happens to be applying commutativity. Baroody and Ginsburg (1986) explain children's responses to such problems in a similar way:

> "... They can choose not to respond, but this is a sure sign of low intelligence. Children are not trained to respond: "An answer is not possible". Indeed because children are usually trained to believe that there must be a correct answer, they may override their intuition that an answer is impossible and manufacture an answer."
> (Baroody and Ginsburg, 1986, p. 103).

Ashlock (1982) also provides such a reason for why children generalize and learn patterns of error:

"... Yet all too often, especially when taught in groups, children do not have prerequisite understandings and skills they need when introduced to new ideas and procedures. When this happens, they want to please the teacher (or at least survive in the situation); so they tend to 'grab at straws'". (Ashlock, 1982, p. 6).

Why do children say '2 - 4 = 0'?

The children who give "0" or "nothing" as answers to problems like 2 - 4 do not all have the same reason for their answers. In reply to "Can you show me how you would subtract 4 from 2?", some of them revealed that there is nothing left to subtract *from* after subtracting 2 (of the 4) from 2; some explained that there is *nothing* left after subtracting 2 (of the 4) from 2 and hence the answer is 0. Most of these children reason like Su:

> Su: (Pause) I took 4, I take away. No! I took 2 (showing 2 fingers), from that I (pause). I have to minus (pause). I took 2, from that I took 2 away; there's 0 left.

Another child who reasoned like Su exclaimed "None left" while trying to work out 2 - 4 on her fingers. These children's answers are quite reasonable, since they realise that they cannot go beyond zero.

For some of the other children who say "nothing" or "0", it is another way of saying "I don't know" or "I don't know what's happening here". Yet other children mean to say "It cannot be done".

Given that these children do not have a concept of negative numbers, the above responses are all reasonable.

Can commutativity be taught?

There is not enough empirical evidence to give a definite answer to this question (in the interviews, only one possible method of teaching was used). As a result of the studies, it can be concluded that it is very difficult for children to learn the concept from *abstract examples only*. Note that in the studies, children were not being given any guidance or feedback. They used icelolly sticks for counting if they wished. With over 40 of the subjects in the two studies not having a complete understanding of the concept, only 5 subjects started applying the concept as a result of solving a sequence of examples (ignoring those who had forgotten the concept, and remembered it after 2 or 3 examples).

We know that children learn the concept of commutativity. We also know that the concept develops gradually over a long period of time (Denvir and Brown, 1986; Hennessy, 1986). There is evidence of children without any formal teaching knowing the concept (Baroody et al., 1983). Following from this, I believe that children, left on their own, without any teaching of the concept of commutativity, with their everyday experiences with concrete examples, will come to a stage where they recognize the concept. I propose that this recognition process can be accelerated by teaching. In addition, teaching can facilitate the *abstraction* of the concept. Teaching should be in a non-abstract, non-mathematical way and should build on children's concrete experiences. Children can be led into a discussion about their experiences. For example, "if you ate 3 smarties first and then 2; and if you ate 2 smarties and then 3, ... Would you eat the same number of smarties in both cases?". The job of the teacher is to stimulate them to think about the concept, and to facilitate the understanding of the concept by relating the non-abstract, non-mathematical experiences to abstraction.

The empirical results presented above have implications for teaching commutativity of addition and non-commutativity of subtraction. For each

type of problem there are a limited number of different answers and a limited number of different strategies. For example, for subtracting a larger number from a smaller one, there were four different answers, as shown in Table 3-3; for the second of a commutative pair of problems, children's strategies could be broadly classified into two: either copying the answer to the first problem or calculating the sum (calculating the sum can of course be analysed further in terms of the counting strategies used, etc.). Using knowledge of the different answers and different strategies and the knowledge of the implications of these for a student's state of knowledge, a teacher (or a tutoring system) can diagnose the student's current knowledge state. Teaching then depends on this diagnosis - designed to help the student progress from his/her current performance level to the desired level.

Associativity

The subjects used one of the following two strategies in solving 3-addend problems (like 4 + 8 + 4):

i) They performed the operations in any order. They *grouped* any 2 of the 3 numbers, and carried out the appropriate operation on them first. Hence, in a problem like a + b + c, they did either a + b, or a + c or b + c first (whichever was easier). For example, for 3 + 5 + 5, children using this strategy, did 5 + 5 first because most children knew doubles, or it is easier to add on a small number at the end or 10 is easier to work with than 3 or 5.

ii) They performed the operations from left to right (linear strategy). That is, they always did the operation on the first 2 numbers to begin with, and then performed the next operation on the result and the last term. Thus, 4 + 8 + 4 would be solved as:

$$4 + 8 = 12$$

12 + 4 = 16

Table 3-4 shows a comparison of the above strategies used by children who used commutativity (stages iii and iv) and those who did not (stages i and ii). It shows that knowledge of commutativity and use of grouping are strongly related. Not knowing commutativity implies not using grouping. In addition, the table reveals that knowing commutativity is a necessary but not sufficient condition for using grouping.

Table 3-4. Relationships between strategies for 3-addend addition and knowledge of commutativity

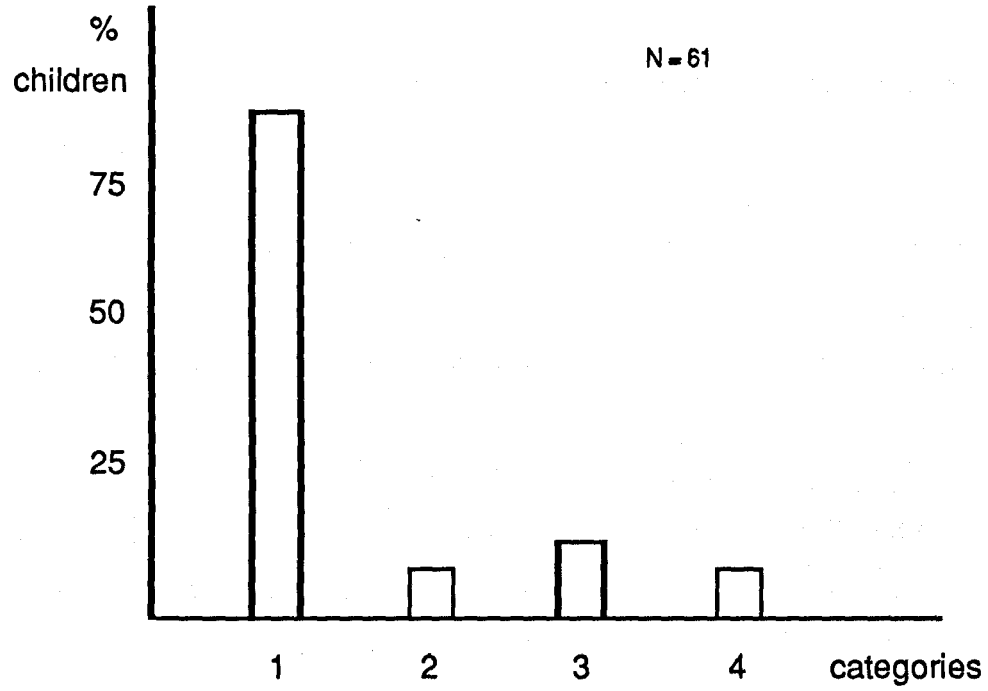
	strategy i (grouping)	strategy ii (linear)
Knew commutativity	50	11
Did not know commutativity	1[1]	15

The results of Table 3-4 show that of the 61 students who knew commutativity, 82% of them used grouping on the 3-addend problems. These students also gave descriptions that showed explicit knowledge of associativity. For example,

> "You have just changed the order of the sum"
> "It doesn't matter what you do first, it's the same thing".

[1] Details of this child's performance is discussed later in this section.

Children seem to progress through several stages in the transition from commutativity to associativity. Figure 3-1 displays the proportions of children who knew commutativity, at different stages of performance on 3-term problems.



categories:

1 - Children who use grouping.

2 - Children who showed explicit knowledge of grouping, but did not use it.

3 - Children who applied commutativity to the first 2 numbers of the 3-addend problems.

4 - Those who did not reveal any knowledge of commutativity in their performance on 3-addend problems.

(note: 2,3,4 represents children who used the linear strategy and fell in one or more of the categories 2, 3 and 4)

Figure 3-1. Application of commutativity to 3-addend problems

There were 3 children who did not use the grouping strategy, but showed explicit knowledge of the concept:

PKP revealed this by not computing the answer to the second of these problems again:

$6 + 8 + 2 = 16$ $(8 + 6 = 14, + 2 = 16)$

$2 + 8 + 6 = 16$

When asked how he had done this, he replied "You repeated the question".

MSN: $3 + 2 + 3 = 8$ $(3 + 2 = 5, + 3 = 8)$

When asked for any other way of doing this problem, she replied "Can also do 3 + 3 first and then add 2".

Empirical evidence of children who appeared to lack understanding of a concept on one task and showed performance consistent with the concept on another task has been provided by Gelman and Gallistel (1978). This implies that children do not always use their conceptual knowledge. Some possible reasons for this are:

i) they do not think of the concept at the time. For example, in case of commutativity, children made comments like:

"I knew it but I calculated it again because I didn't think of it".
"How stupid of me; I didn't think of it".

ii) even if they are aware that a particular concept is applicable, they know that after all they will get the same answer whether they use it or not. For example, there were a number of children in the commutativity experiments who knew the concept and were computing the answers to the second of the commutative pairs of problems. They were doing this so that they could check their

answers to the first problem. In general, children do not always use the most efficient strategy in their problem solving (Luchins, 1942).

iii) they have not reached the stage in the development of the concept where it can be applied.

Luchins (1942), in one of his experiments, gave 11 of his subjects (aged nine to fourteen years) a series of problems to solve. All the problems could be solved by the same strategy, but the seventh and eighth ones could be solved using a more direct (simpler and faster) method. However, not one subject used this more direct method. Having become habituated to the method of solution that worked for the first six problems, they used it in the succeeding similar problems. Later when they were shown the more direct method, the subjects spontaneously made comments like:

"How dumb I was"
"How blind I was".

Luchins also found that children sometimes wanted to repeat a method throughout. Some of the comments made by his subjects are:

"why should I bother seeking new methods"
"I did the best I could do and, after all, my answer was correct"
"It's senseless to do the same thing many different ways".

This shows that some children consciously decide not to search for alternative methods once they have one that works.

There were 3 children who knew the concept of commutativity, but did not extend it to 3-addend problems. Their inability to transfer the knowledge of commutativity implies that there is a stage between knowledge of a concept and its application. If it was not for this stage (revealed by these 3 children), commutativity would be a necessary *and sufficient* condition for grouping.

Eleven of the older subjects, who used grouping on 3-addend addition, and who were familiar with the four operations, were given 3-addend problems with different operators besides those of addition. (Appendix 4 marks these subjects). All the subjects were given the same set of problems (listed in Appendix 5).

The results are listed in Appendix 6. None of the subjects used grouping on the problems with a combination of operators. Four of the subjects used grouping on the addition-only and on the multiplication-only problems, and the linear strategy on the subtraction-only, division-only and combinations-of-operators problems. One subject generalized her grouping strategy to all problems with 1 type of operator only; hence, 12 / 6 / 2 = 12 / 3 = 4, 16 - 10 - 5 = 16 - 5 = 11. The rest of them used grouping on the addition-only problems and the linear strategy on the other problems.

An interview with the child (PD) who did not know commutativity, and yet seemed to apply associativity (see Table 3-4), revealed that she did not have any knowledge of the latter concept either. She was using the labour saving strategy, COL, while doing the commutativity problems. For the problem, 7 + 13, she started counting on from 13. Then, for 13 + 7, she repeated the counting. When asked explicitly if she was aware that the order in which 2 numbers are added does not make a difference to their sum, she replied "No". On the 3-addend problems, she began from the largest addend. For example, 3 + 3 + 12 - she started from 12 "because 12 is the largest number". This strategy seems to be an extension of the COL strategy that she used on the 2-term problems. This child is an example of one whose algorithms could be said to embody the concepts, but she does not explicitly possess that knowledge. PD's behaviour highlights the need to distinguish between the knowledge of a concept and the use of algorithms that presuppose the concept. This type of distinction has also been discussed by other researchers (Baroody, 1984; Hennessy, 1986; Resnick, 1983).

3.3.3 Conclusions

Different levels of performance of the concept of commutativity have been identified. However the complex interrelationships between the sub-levels need to be studied in more detail.

In the studies reported in this chapter, commutativity of one operation only (addition) has been considered. One would expect variations in the results had the other operations been considered as well. Another dimension of variation would be the type of number system (integers, fractions, decimals, etc.).

The studies suggest that the acquisition of associativity and that of commutativity are interrelated. There was evidence of children who were at stage iii), for example using commutativity for small numbers only, and were using grouping on 3-addend problems. There were also children who were noted for applying commutativity to 3-addend problems (partially - only to the first 2 addends). There was not a single child who was noted using grouping (commutativity or associativity) on 3-addend problems and not using commutativity on 2-addend problems. This shows that commutativity is a necessary prerequisite for the acquisition of associativity.

The evidence of use of procedures independent of their conceptual knowledge, and that of existence of conceptual knowledge independent of their applications, highlights that there is an important stage of linking conceptual and procedural knowledge.

3.4 Longitudinal Study

The aims of the study were as follows:

i) to study the development of children's understanding of commutativity and associativity and of their addition strategies at a more detailed level and

ii) to double check the results obtained in the earlier study.

3.4.1 Method

Twelve children (7 boys and 5 girls) aged between four and eleven were interviewed seven times over a period of 1 year and 8 months. The time intervals at which the interviews were carried out are shown in Table 3-5. The subjects were from the same local school as that in the pilot study. Six of them were from the pilot group. The subjects were at different levels of understanding of commutativity and associativity - they had different lengths of school experience. Originally, the study included 15 children - of the 3 of them who were dropped, 2 were absent quite often and 1 had left school during the course of the study. The materials and the procedure for the interviews were exactly the same as those in the main study above.

3.4.2 Results

The results of each child's interview are presented below. The ages are given in brackets and are as they were at the first interview. Table 3-5 summarises the results.

1. Samuel (6 years)

At his first interview, Samuel showed no signs of knowledge of commutativity. He used the 'count all from the first addend' (CAF) strategy for solving the addition problems. For the second of a pair of commuted problems, he swapped the set of counters and recounted. For example, for the problem 4 + 7, he put 4 counters on the table to his left, and 7 counters to his right and counted them all to get 11 as the solution. For 7 + 4, he

dragged the set of counters on the left to the right and the set on the right to the left and recounted. When asked if 8 + 3 would give the same answer as 3 + 8 or not, he replied: "Different, because they are the other way around."

At his second interview, Samuel sometimes used the 'count on from the larger addend' (COL) strategy for addition. He still used counters, but did not pay particular attention to addend order. He copied the answer to the first of a commuted pair of problems. Furthermore, he made an explicit statement of commutativity: "... because it's the same numbers ...". For 3-term problems, he applied the grouping strategy, and for subtraction problems (like 2 - 5), he explained (logically, reasonably and with understanding of subtraction) why they could not be solved.

From his third session onwards, Samuel used COL and grouping strategies consistently. He also showed understanding of commutativity and associativity.

2. *Freda (6 years, 7 months)*

For the first two sessions, Freda used CAF and 'count on from the first addend' (COF) strategies. She did not indicate any knowledge of commutativity. All throughout the study, she applied 'smaller from larger' strategy to subtraction problems. From her third session onwards, she used COL strategy for 2-term problems and showed explicit knowledge of commutativity. She also used grouping.

3. *Grenville (8 years, 8 months)*

Grenville used CAF for all the addition problems and showed no knowledge of commutativity in the first session. In his third session, he used COF and showed some knowledge of commutativity (e.g. when asked whether the answer would be the same or different ..., he replied 'same'), but did not

use it in his solutions. In his fifth session, he copied the answer to the previous problem, but not always; he generalized commutativity to subtraction and carried on using COF. In the last two sessions, he used 'count all from the larger addend' (CAL) and COF and generalized to subtraction. In the last session, he used the same strategies except counted on fingers instead of using counters. For 3-term problems, he always used the linear strategy.

4. Craig (6 years, 4 months)

In the first two sessions, he did not use commutativity but treated every problem in the same way. He did not know how to solve 3-term problems until the fifth session, when he used the grouping strategy. In the third and fourth sessions, he used CAF. In his fifth session, he used commutativity, CAL, grouping, and 'smaller from larger' strategy. In the sixth session, he showed explicit knowledge of the concepts. In the last session, Craig still used 'smaller from larger' strategy and used COL.

5. Steven (7 years, 9 months)

From the beginning of the study, Steven used commutativity, i.e. he copied the answer to the previous problem. He also showed knowledge of the concept: "this is the same as that", and used the grouping strategy. In his first two sessions, he used CAL and 'smaller from larger' strategy. After that, he used COL and informal strategies embodying commutativity, recognized the similarity between two associative problems and said "cannot be done" for subtraction problems.

6. *Susan (5 years, 11 months)*

Susan knew the concepts and used COL and grouping throughout the study. She was not ready for subtraction, so tried generalization and 'smaller from larger' strategy to reach an answer.

7. *Jaimy (5 years)*

Jaimy used CAF and linear strategy at all the five interviews. She did not copy the previous answers in the cases of commuted pairs of problems, and did not show any signs of knowledge of the concepts. For subtraction, she replied "can't do them" or "none".

8. *Daniel (5 years, 9 months)*

All throughout, Daniel used CAF strategy. He did not know how to do 3-term addition and said "none" to the subtraction problems. He did not show any improvement over the period of the study.

9. *Sundip (5 years, 6 months)*

Throughout the study, Sundip's only strategy transition was from CAF to COF. In the first five sessions, he used CAF. In the sixth session, he used CAF and COF. He did not reveal any knowledge of commutativity, nor any sign of improvement on the concept. For 3-term problems, he used the linear strategy throughout. For subtraction, he either said "zero" or "none".

10. Kelly (3 years, 11 months)

Kelly showed knowledge of commutativity, copied the answer to the previous problem, but still used CAF for the first four interviews. For 3-term problems, she used the linear strategy. On the last three sessions, in addition to CAF, she used COL. On the final session, she used COL and grouping. For the subtraction problems, she said "can't do it" or "don't know".

11. Karina (5 years, 4 months)

Karina used COL and grouping throughout. She showed explicit knowledge of commutativity and used it in all the sessions. For subtraction, she said "none left".

12. Lisa (5 years, 5 months)

For the first four sessions, she used CAL and CAF, and linear strategy. She copied the previous answer. From the fifth session, she used grouping and COL. For subtraction, she either said "none" or did 'smaller from larger'.

Table 3-5 is a summary of the subjects' development of 2-term and 3-term strategies during the course of the longitudinal study. Cells marked '-' indicate that the child was not interviewed on that particular day.

Table 3-5. Summary of strategy development

Time (weeks)	0	10	47	48	51	63	85
Samuel	CAF	COL copies grouping	COL grouping	COL grouping	COL grouping	COL grouping	COL grouping

Freda	-	CAF COF	COF not copy	-	COL grouping	COL grouping	COL grouping
Grenvill	CAF	CAF	COF linear (knows assoc.)	COF copies answer but not always linear	COF copies answer linear	CAL COF linear	CAL COF linear
Craig	CAF not copy	CAF	CAF does not know 3-term	CAF does not know 3-term	CAL grouping	CAL grouping	COL grouping
Steven	CAL	CAL copies linear	dis-regards addend order - uses facts of 10 grouping	COL informal methods grouping	COL informal methods grouping	COL grouping	COL grouping
Susan	COL grouping	COL grouping	COL grouping	-	COL grouping	COL grouping	COL grouping
Jaimy	CAF linear	-	CAF linear	CAF not copy linear	CAF linear	CAF not copy linear	-
Daniel	CAF not copy	CAF not copy	CAF not copy	CAF does not know 3-term	CAF does not know 3-term	CAF does not know 3-term	
Sundip	CAF not copy	CAF not copy	-	CAF	CAF linear	CAF COF linear	CAF COF linear
Kelly	CAF copies ans	-	-	CAF copies ans	CAF COL linear	CAF COL linear	COL grouping
Karina	COL grouping	-	COL grouping	COL grouping	COL grouping	COL grouping	COL grouping
Lisa	CAF copies	-	-	CAL linear	CAL copies	CAL copies linear	COL grouping

3.4.3 Discussion

The results confirmed the following observations reported in the main study:

i) Children do not always use the knowledge they possess. They do not always use their knowledge to shortcut their computational effort. This is demonstrated by Kelly, for example, who knew commutativity but was not using the COL strategy. This finding has also been reported by Baroody, Ginsburg and Waxman (1983), Carpenter (1986) and Hennessy (1986).

ii) Children do not always use the most efficient strategy that they can. For example, Samuel did not use COL all the time. The study also supports Fuson's (1982) finding that children use a variety of strategies.

iii) All the strategies observed in this study are contained in the 'space' of observed strategies in the main study, i.e. no new strategy was discovered in this study.

iv) The same stages of development of the concept of commutativity were evident. Each child in the study, who showed any sign of knowledge of the concept, could be categorised into one of stage ii), iii) or iv) in the levels of development of commutativity discussed in section 3.3.2 above.

v) The subtraction results were similar to those obtained in the main study. Children gave similar answers to subtraction problems as those in the main study. In the analysis of those results, one of the explanations proposed for children's generalization of commutativity to subtraction was that they have been led to do so by the sequence in which the problems were presented. The longitudinal study gave support to this explanation. There were children who did not generalize on subtraction problems when they were presented at the beginning of the interview session but did so when they were presented after a sequence of commutative addition problems.

Note that the problems were commutative pairs (e.g. 4 - 2, followed by 2 - 4). This might have led them to generalize more than if they were presented with a single subtraction problem (e.g. 2 - 4). This can be explored as further work, which could involve an investigation of children's responses to a single subtraction problem, and a detailed study of the effect of priming on children's responses.

vi) As can be seen from Table 3-5, each occurrence of 'grouping' is accompanied by strategies that either start counting from the larger addend (i.e. COL and CAL) or disregard addend order (e.g. Steven). This result confirms the conclusion reached from the main study that knowledge (at least procedural) of commutativity is a prerequisite for the knowledge of associativity. The table also shows evidence of children who used strategies that start counting from the larger addend (e.g. Kelly), but still use the linear strategy. Once again, this confirms the hypothesis from the main study that there is a transition phase from 2-term 'counting from larger addend' strategies to grouping strategy on 3-term problems.

The above results support previous work on the general trend of transition of strategies towards more efficient ones (e.g. Fuson, Secada and Hall, 1983; Resnick and Groen, 1977). Resnick and Groen's (1977) study suggested that such transitions take place without instruction; children invent them for themselves. The substantial literature on informal methods supports this too (Carpenter and Moser, 1983; Resnick and Ford, 1984; Starkey and Gelman, 1982). From this study, one can conclude the following possibilities for transitions of strategies:

i) Agreement with Resnick and Groen's suggestion that children invent efficient strategies for themselves.

ii) Children who were interviewed could have had discussions with each other, which could have influenced their performance.

iii) The transitions are influenced by children's practice at solving such problems at school (partial disagreement with Resnick and Groen).

Furthermore, the study revealed that children may use a particular strategy for a long time before a transition to another strategy takes place. For example, Sundip used CAF for a year before he started to use COF.

To conclude, the longitudinal study showed that there is some effect of time on the development of strategies for addition. Over the 20 month period, there was no significant effect on their subtraction strategies. Furthermore, transitions to more efficient strategies were observed. The study also showed that once a child knew and used the concept of commutativity and associativity, s/he subsequently used it over the duration of the longitudinal study. Finally, knowledge of the concepts of commutativity and associativity are interrelated, and there is a developmental trend: complete knowledge and application of associativity is followed by knowledge of commutativity.

3.5 Educational implications

The levels of performance of commutativity identified in the studies above could be used to design tasks to facilitate the learning of the concepts of commutativity and associativity. The tasks can be designed to proceed structurally through the performance levels. Children's textbooks suggest that standard classroom teaching does not guide children clearly towards the learning of the concepts.

Existing literature and evidence from the studies reported in this chapter show that children's conception of commutativity and of subtraction leads to generalization errors on problems where the minuend is less than the subtrahend. Even children who are old enough to know negative numbers may generalize commutativity on such problems. This suggests that at an

earlier age, some explanation of such problems in relation to commutativity should be given to students. At the stage when children apply the concept of commutativity to addition, problems like 4 - 2 and 2 - 4 ought to be introduced to show that the concept does not apply to subtraction. Such an introduction does not need to be an introduction to negative numbers.

Subtraction problems can be introduced with addition for teaching the concept of commutativity. For example, after going through a sequence of positive examples of the concept, i.e. addition problems, when a problem like 2 - 4 is introduced, the child is puzzled. Such a 'puzzled' state could provide an inquisitive mind with a good environment for learning. Those children who generalize commutativity to subtraction and are given feedback on their responses, need to 're-think' which could lead to an improvement on their understanding of the concept. Such situations could be created to teach commutativity. Furthermore, when they know commutativity for addition, such subtraction examples may help to avoid future generalizations of commutativity to subtraction.

On subtraction problems like 3 - 7, those children who respond reasonably, such as "how can you take away ... 3 is smaller than 7 ..." have already made the distinction that subtraction is not commutative. Those who do not make the distinction could be provided with further guidance.

3.6 Summary

This chapter reported three studies that were carried out to investigate children's acquisition of the concepts of commutativity and associativity. The main conclusions of the study are as follows:

i) The studies provide evidence of the need to make a distinction between conceptual and procedural knowledge of commutativity and associativity.

ii) A space of strategies for solving subtraction problems, 2-term and 3-term addition problems has been identified.

iii) The studies indicated that knowledge of associativity follows from that of commutativity.

From the results of the studies reported above, production-rule models of children's strategies for solving arithmetic problems like 5 + 6 and 8 + 5 + 9 have been implemented. The goals of the modelling were as follows:

i) to understand and clarify the details of children's performance and

ii) to explore the potential application of the models in an ITS.

The following chapter is an account of the program, which simulates children's strategies at different stages of development using production rules.

Chapter 4

PRODUCTION-RULE MODELLING

Based on the results of the study reported in the previous chapter, models of children's strategies for solving 2- and 3-term addition problems have been implemented. There are two main reasons for the modelling. Firstly, the models can be used in an ITS. Secondly, it allows a detailed analysis of children's performance. This chapter describes the program, PALM, which simulates the strategies using production rules. PALM stands for "Production-rule Arithmetic Learning Modeller". There are two major parts to PALM: one for production-rule modelling and the other for modelling learning. The first part is discussed in this chapter, and the second in the next chapter. This chapter begins with an introduction to production systems, followed by a review of three examples of production systems. Then the development of PALM is discussed, concentrating on its components, and on the features of the representation and their functions. Simulations of strategies and the procedure for calculating estimates of their efficiencies are described. Finally, the production-rule models are discussed in relation to the data.

4.1 Production systems

In this section, the basic components of a production system are outlined. This is followed by brief descriptions of three production systems.

4.1.1 A brief description

Production systems consist of three basic components:

i) Working memory - a set of elements which represent the results of rule firings and the current status of the system.

ii) A set of production rules - these constitute the operational knowledge representation component of a production system. Production rules have a left-hand side, which represents the conditions under which a rule may be applied, and a right hand side which states the actions to be performed when the left-hand side conditions are satisfied.

iii) Interpreter - the driving engine of the production system. It matches the rules with the working memory items, selects a rule, fires it and updates the working memory.

A rule is of the form

condition(s) ---> perform action(s)

where 'conditions' represent a specific state in the problem-solving process and 'perform action' represents the step in the solution that is carried out at that particular state. The results of the actions are stored in working memory; the working memory is updated after each action. The updated working memory shows the stage in the problem solving process that has been reached. The state of the memory is matched against the condition sides of the production rules in order to select the next rule to 'fire'. At this stage, more than one rule could match the memory items. Production systems have strategies called conflict resolution strategies to select one of the possible rules. Some of these include:

Refraction - once executed, an instantiation of a given rule may not be executed again, i.e. a rule which matches to the same working memory items as it did previously is eliminated.

Recency in working memory - rules which match with those items in the working memory that have been asserted more recently are selected over those matching older working memory items.

Recency in rule memory - select the most recently created rule(s). This strategy is only applicable in systems that are able to learn new rules.

Rule ordering - select the first of the list of possible rules.

Specificity - select a rule which is more specific than other rules. Specificity can be measured in a number of ways. One of these ways is to measure the complexity of the condition side of the rule, and the preferred rule is the one that is maximally complex. Another measure of specificity is to delete from the conflict set instantiations of those rules whose condition sides are proper subsets of the condition sides of other instantiated rules. A third way compares the sets of working memory elements that match the conditions of the rules in the conflict set. If the matching working elements of one rule are a subset of the matching elements of another, then the first rule is ruled out.

Random selection - choose a rule at random.

The cycle of matching rules against working memory items, performing the conflict resolution, 'firing' the selected rule and updating the working memory continues until either there are no more matching rules or a stopping condition is satisfied.

4.1.2 Examples of production systems

Described below are three production systems that are used for cognitive modelling. They are described with respect to the three components, working memory, production memory and the interpreter.

GRAPES

GRAPES, as discussed by Brownston, et al. (1985), was constructed for goal-directed problem solving. It has two distinct memories, working memory and goal memory. The working memory is similar to those of other production systems. It contains facts that are used by rules to make inferences. The goal memory contains goals or tasks, and is stored as an 'and/or' tree of subgoals. An 'and' branch requires all the subgoals to be satisfied, and an 'or' branch requires only one of them to be satisfied. At a general level, the system's processing is directed by the goals in the goal memory. At a more detailed level, the goals or tasks are fulfilled by executing rules which depend on the working memory elements. Each rule in GRAPES has a group of goal parameters and a group of tests. The system first matches the goal parameters of its rules against the current goal, and then matches their tests. The condition sides of the rules can have one or more of three types of tests: goal tests, working memory tests and function tests. Goal tests can test against goals other than the current goal. Working memory tests are similar to those in other production systems. Function tests are LISP predicates that return a true value if the predicate succeeds. The conflict resolution strategies, in the order in which they are performed are refraction, recency, specificity, and arbitrary choice.

PRISM

PRISM (Langley, 1983) stands for Program for Research Into Self-Modifying systems. It has a long-term memory and a working memory. The elements in the working memory have an associated activation. The activation is a measure used for representing forgetting. At every cycle, PRISM examines each working memory element to see if its activation has fallen below a user-specified threshold. When this occurs, the element is removed from working memory and is forgotten. The long-term memory contains these forgotten elements. They have associated trace strengths,

which are used in directing spreading activation, the process for retrieving elements from long-term memory and adding them to working memory. Rules in PRISM may contain negated conditions. Each rule has an associated strength, which is used during conflict resolution. The conflict resolution strategies are rule ordering and refraction. The order of rules in the conflict set is determined by options like rule strength and the recency of the productions. The refraction strategy eliminates all instantiations that applied on the previous cycle.

OPS5

OPS5 (Brownston et al., 1985) is one of a family of OPS production systems. In OPS5, working memory elements are represented as attribute-value elements. Each item has its class, followed by the name of the class, and then pairs of attribute names and values. A prefix operator (! in the example below) is used to distinguish attributes from values. The following is an example of an attribute-value element:

```
(Person      !name Kate !mother Helen !father Jim !Age 7)
```

Each working memory element also has a time tag or a recency value which indicates when the item was entered, or last modified. Rules in OPS5 are of the form: production name followed by a set of condition elements, an arrow and then a set of actions. The conditions specify patterns that are to be matched against working memory elements. When dealing with numbers, OPS5 conditions can have operators like =, <, >, <=, >=. In addition, conditions can have disjunctions and negations. The right-hand sides of the rules consist of one or more of twelve predefined action types, for example, 'make', 'modify', 'remove', 'halt', 'write'. The actions can also contain functions, e.g. 'compute'. The conflict resolution strategies, in the order of their application, are refraction, recency, specificity and random selection.

The main application of GRAPES and PRISM is in cognitive modelling. OPS5 was not intended as a serious model of the human cognitive architecture (Neches, Langley and Klahr, 1987). The special feature of GRAPES is the separate goal memory. PRISM and OPS5, on the other hand, do not make a distinction between facts and goals and store them together. OPS5 is more efficient than GRAPES and PRISM. All three architectures also have learning components.

In the next section, PALM, a production system implemented for the specific purpose of modelling children's arithmetic strategies, is described.

4.2 Implementation of PALM

An existing production system, like one of those described above, could have been used instead of implementing an interpreter. Instead, information from such systems was used and an interpreter was implemented to serve the specific purpose of this research. Some of the reasons for implementing a specific one are as follows. Firstly, using an existing interpreter means that one is restricted to the representation language of that interpreter. As ideas develop, it is easier to extend one's own interpreter accordingly. Secondly, PALM includes a learning component, and it is easier to build a 'simple' interpreter rather than adapt a more complicated existing one to incorporate the type of learning that is modelled in PALM. PALM learns by generalization and by condition learning. PRISM includes learning by generalization, but its mechanism for generalization is significantly different from the learning mechanism in PALM. OPS5 does not contain any mechanism for learning by generalization. A discussion on learning issues is postponed until chapter 5.

PALM was first implemented in Interlisp-D on a Xerox machine. It was subsequently transferred to a Macintosh SE/30 and the code translated into Common Lisp.

4.2.1 Working memory

The working memory is represented by a global variable, and is initialized to 'nil' before the program is run. The program takes an example problem as input. The problem is added to the working memory to initiate the 'firing' of rules. As the interpreter performs the actions of the rules, items are added to the working memory. The working memory is a list containing its elements in embedded lists. The following is an example of the structure of the working memory:

(... (col 2 5 1 4) (number 1 4) (number 2 5))

The last two items represent the problem, 4 + 5, which have been added to the working memory before any rule 'fired'. The first item is a result of a rule that fired for solving the input problem using the COL strategy. The working memory is updated by adding items at the front, i.e. the left-hand end of the list.

4.2.2 Productions

In PALM, a rule is in the form of a list consisting of three elements.

i) A list containing one or more patterns, where each pattern is a condition.

ii) An arrow that separates the condition from the action.

iii) A list containing one or more patterns to be added to the working memory, where each pattern is an action.

The following is an example of a production rule in PALM:

((number =I1 =X) (number =I2 =Y) (not (used =I2 =Y)) (not (used =I1 =X)))
---> ((col =I1 =X =I2 =Y))

which represents the problem x + y and the COL strategy. The rules represent problem-solving strategies. Conditions can have negated items, prefixed by 'not', as demonstrated in the above example. Apart from negated clauses, the conditions are in such a form that testing against working memory items involves simple pattern matching. The actions contain clauses which are like those in the conditions (except for negations). These clauses are added to the working memory directly, with the variables instantiated. Actions can also be functions, which are evaluated and their results are added to the working memory.

4.2.3 Interpreter

The three main functions of the interpreter are matching, conflict resolution and rule execution. In the process of solving a given problem, the interpreter performs the cycle of matching rules against working memory items, selecting a rule and 'firing' it, until either an answer is reached, or there are no more rules to 'fire'.

matching

The matching process applies the production set to the working memory and returns a list of the instantiated 'action' parts of the rules. We shall describe the matching process for one rule using the following simplified example:

conditions: *((number =X) (number =Y))*

working memory: *((number 7) (number 15) (number 6))*

Each condition item is matched against the working memory items to get a list of bindings for the variables in the condition. For the first condition item, for example, the result of the match is as follows:

((3 (=X 7)) (2 (=X 15)) (1 (=X 6)))

There are three bindings for the first condition, i.e. the condition matches all three items in the working memory. The integer in front of each list of variable bindings shows the position of the element in the working memory to which the condition is matched. It indicates the recency, relative to other bindings, of the matching working memory element. The larger the integer, the more recent the element. This information is used in conflict resolution (described below). For the second condition, the match is as follows:

((3 (=Y 7)) (2 (=Y 15)) (1 (=Y 6)))

At this stage, the two sets of bindings of the conditions are merged, to get a list of consistent bindings. This process returns nine possible matches for the condition side of the rule:

[(3 3 (=X 7) (=Y 7)) (3 2 (..) (..)) (3 1 (=X 7) (=Y 6)) (2 3 (..) (..)) (2 2 (=X 15) (=Y 15)) (2 1 (=X 15) (=Y 6)) (1 3 (=X 6) (=Y 7)) (1 2 (=X 6) (=Y 15)) (1 1 (=X 6) (=Y 6))]

PALM's next step in the matching process is to eliminate those matches where more than one variable in the conditions is bound to one constant in the working memory, e.g. '6' is bound to 'x' as well as to 'y'. Note that in problems like 6 + 6, the two 6's are treated as two constants. This is detailed in section 4.3.2. The recency values in the above list reveal whether the same constant has been bound to more than one variable or not. *(1 1 (=X 6) (=Y 6))*, for example, implies that 'x' is bound to the first item in the working memory (6) and 'y' is bound to the same item as well. Eliminating such redundancies, the list of matches is as follows:

[(3 2 (=X 7) (=Y 15)) (3 1 (=X 7) (=Y 6)) (2 3 (=X 15) (=Y 7)) (2 1 (=X 15) (=Y 6)) (1 3 (=X 6) (=Y 7)) (1 2 (=X 6) (=Y 15))]

The handling of negated conditions is discussed in section 4.3.3. The matching process described above is repeated for every rule in the production set. The list of successful matches for all the rules is then passed through the conflict resolution strategies in order to choose one rule to 'fire'.

Conflict resolution

The interpreter applies the following conflict resolution strategies:

i) Recency - select the rule instantiation which matches to the most recent item in working memory.

ii) Rule ordering - if there is more than one applicable rule instantiation after applying the first resolution strategy, then select the first of these.

If we assume that the rule in the above example is the only one that is instantiated, then the conflict set is the list of instantiations of that rule:

[(3 2 (=X 7) (=Y 15)) (3 1 (=X 7) (=Y 6)) (2 3 (=X 15) (=Y 7))
(2 1 (=X 15) (=Y 6)) (1 3 (=X 6) (=Y 7)) (1 2 (=X 6) (=Y 15))]

Applying the recency strategy, the conflict set reduces to:

[(3 2 (=X 7) (=Y 15)) (2 3 (=X 15) (=Y 7))]

Applying the rule ordering strategy, the first instantiation is chosen. Hence the appropriate action corresponding to the condition, *(number 7) (number 15)*, is carried out.

Note that the conflict set is already reduced by not including those instantiations of rules whose actions result in elements that are already present in the working memory. Hence, a sort of refraction is applied at the

matching phase. The main purpose of this was to avoid the continuous instantiation of a rule to the same set of working memory items.

4.3 The representation

The models are intended for representing students' problem-solving strategies in an ITS so that it can then offer better alternative strategies. One of the reasons for choosing a production-rule formalism is this application of the models. In an ITS, the tutorial goals can be represented as production rules as well, where the left hand sides of the rules would be the models representing students' strategies and the right hand sides would be the appropriate tutoring actions. The student's input is compared with the left hand sides of the rules. In addition, production systems can be 'run' in order to make predictions of a student's possible outcomes. In our case, for example, if the system is 'run' on the problem, 3 + 4 + 5, the system returns the solution using either the grouping strategy or the linear strategy. Such outputs can be used to predict students' solutions and strategies.

Furthermore, this formalism was chosen because of the ability of production systems to learn new rules. The structure of production systems allows additional code for learning to be incorporated with the existing system without affecting its previous behaviour. The modularity of rules in production systems makes it easy for the system to continue to work if rules are added to or deleted from the existing set of rules. The collection of papers in the book edited by Klahr, Langley and Neches (1987) demonstrates this capability of production systems. PALM's attempt at modelling transition from one knowledge state to another is discussed in Chapter 5.

Since PALM includes a learning component as well, this needed to be taken into consideration when choosing the description language. We needed a representation language that describes the example problems, and has the

potential of describing generalizations of examples. The descriptors chosen to describe problem states are *(number A X)*, *(adjacent X Y)* and *(used A X)*, where *(number A X)* represents a number X, whose position in a given problem is represented by A; *(adjacent X Y)* refers to X and Y being next to each other, that is the difference between their positions, A and B, is 1; and *(used A X)* marks the numbers that have been added. For example, the problem 4 + 5 would be represented as:

(number 1 4) (number 2 5) (adjacent 4 5)

A rule for adding two numbers is represented as follows:

(number =A =X) (number =B =Y) (adjacent =X =Y) (not (used =A =X))

(not (used =B =Y)) ---> *(do addition)*

that is, add two numbers that are adjacent to each other, and that have not already been added. The 'do addition' part of each rule consists of a detailed addition strategy.

The above representation was chosen, especially the predicate 'adjacent', so that the models of problem-solving behaviour are restricted to solving 2-term problems only. They are not general enough to solve problems with more than two addends. In order to solve such problems, the program learns to construct new rules. To do this, it needs a representation that captures all the information required for solving 2-term problems, so that it can handle new problems with little restructuring of its existing knowledge.

In the following subsections, we discuss each component of the rules, and the aspects of students' performance that they simulate.

4.3.1 'Used'

When children count with concrete objects, they normally put to one side the set of numbers that they have finished counting, in order to distinguish it from the set(s) that remain to be counted, so that they do not count the items in a set more than once. In production-rule modelling, we need a way to keep a record of this as well, in order to avoid the problem of the rules matching the same numbers. Hence, when numbers are added, one of the actions of the rules is to mark the numbers as they are used. This is done by adding to the working memory the facts that the numbers that have just been added, are now used. For example,

(number =A =X) (number =B =Y) ... (not (used =A =X)) (not (used =B =Y))
--->... (used =A =X) (used =B =Y)

On the condition side, the rule states: "if there are two numbers that *have not* been used" and on the action side: "do the addition and add to the working memory the facts that these numbers *have* been used".

Furthermore, the 'used' clauses provide a means for deciding the state at which a problem has been solved. The rule interpreter, at each cycle of rule firing, checks whether the solution state has been reached. The addends in the given problem are compared with the addends that have been used. When all the addends have been used, and there is an additional (resulting) number, then the problem has been solved.

4.3.2 Indexing

Related to the problem of distinguishing between the used and unused numbers, we also have to distinguish between two or more addends that have the same value (e.g. 3 + 3). This, again, is a distinction that children (perhaps unconsciously) take into account. When counting 3 + 3, with concrete objects or using fingers, they count out **two** sets, one to represent

each 3. When using the 'counting on' strategy, one 3 is used as the number from which to start counting; the other 3 represents the number that is counted on. The two 3's are somehow different. For making this type of distinction explicit in the modelling, each number in a problem is given a unique index (a place holder). Hence, 4 + 7 + 4 is represented as *(number 1 4) (number 2 7) (number 3 4)*. If the numbers are not given a unique index, then after doing 4 + 7, in 4 + 7 + 4, the first 4 is used, and the remaining 4 will never get matched, since the conditions in the rules specify that a number matches only if it is not used, and the two 4's are not distinguished. Indexing marks the 4's as two different ones.

4.3.3 Negation

The condition part of the rules may contain negated items. Negated items are those that are preceded by *'not'*s. Negation is used in rules to provide constraints on the items in working memory that can be matched. A negated condition succeeds if there are no working memory elements that satisfy the condition. It is handled by matching the non-negated condition and adding the result of the match to a global variable (referred to from here-on as *nots*) and comparing it to the matches from the other condition-clauses (referred to as *previous_bindings)* of the rule. *Nots* contains a list of instantiations of items in working memory on which a rule is restricted to fire. *Previous_bindings* is the list of successful bindings to which the rule can fire. After matching each clause, any item that exists in both *nots* and in *previous_bindings* is removed from *previous_bindings*, since it is not consistent with all the condition clauses.

This is illustrated further with the following example, where the left-hand side of a rule has one negated condition and the working memory contains four elements:

conditions: *(number =I1 =X) (not (used =I1 =X))*

working memory: *(number 1 7) (number 2 7) (used 2 7) (used 3 14)*

The first condition instantiates *previous_bindings* to:

((=I1 1) (=X 7)) and ((=I1 2) (=X 7))

The second condition instantiates *nots* to:

((=I1 2) (=X 7)) and ((=I1 3) (=X 14)),

that is, we wish to exclude numbers '7' (with index '2') and '14' as successful bindings of X since they have already been used. Comparing *nots* and *previous_bindings*, ((=I1 2) (=X 7)) is common and is removed from the *previous_bindings* list. Hence, at this stage the set of bindings that is consistent with the two condition clauses is ((=I1 1) (=X 7)). (The conditions required a number that is not used. There is one item in the working memory, *(number 1 7)*, that satisfies these conditions).

4.4 2-term problems

In this section, we describe simulations of children's strategies for solving problems like 4 + 5, that were observed in the empirical studies.

4.4.1 Strategies

The following is the basic rule template for all the strategies for 2-term addition:

(number =A =X) (number =B =Y) (adjacent =X =Y) (not (used =A =X))

(not (used =B =Y)) ---> *(do addition)*

where 'do addition' solves a given problem using one of the observed strategies.

For each strategy, the condition sides of the rules are the same. The 'do addition' part models the different strategies. These are CAF, CAL, COF and COL (discussed in chapters 2 and 3). Note that all strategies are equally applicable - there is no attempt to model choice of strategy at this stage. Each of these strategies also has the alternative, 'copy the answer to the previous problem' (for the case of commuted pairs of problems). This is to model the strategies of those pupils who recognize the similarity of the sums in such pairs of problems and do not compute the answers to the second of the problems.

The following set of rules model the COL strategy:

(number =I1 =X) (number =I2 =Y) (adjacent =X =Y) (not (used =I1 =X)) (not (used =I2 =Y)) ---> (col =I1 =X =I2 =Y)

(col =I1 =X =I2 =Y) ---> (fn coladd =I1 =X =I2 =Y)

(addd =I1 =X =I2 =Y) ---> (fn addd =X =Y) (used =I1 =X) (used =I2 =Y)

where function names are preceded by *fn*. Function *coladd* takes the two addends as arguments and returns *addd I1 X I2 Y*, where *X* is the larger of the addends. This is because the aim of the COL strategy is to start from the larger addend. *Fn addd* computes the sum of the two addends. For example, for the problem, 7 + 9, the first rule fires, and *col 1 7 2 9* is added to the working memory. This result matches the left-hand-side of the second rule, whose action evaluates *fn coladd* to *addd 2 9 1 7*. At this stage, the third rule fires, evaluating *fn addd* which adds the *7* to the *9* and also outputs the efficiency of the *coladd* strategy. The computations for efficiencies of the different strategies is discussed in the next section (4.4.2).

The internal details, that is the one by one counting of each addend, in each strategy has not been modelled using rules. This is because it is not

important for the purpose pursued here, which is to study the transition from ability to solve 2-term problems to that of solving 3-term problems. Furthermore, the strategies are modelled at a level where they can be differentiated from the other strategies, and this has been done by focussing on which addend is considered first, and then noting whether the counting begins at '1' or at one of the addends. The counting details of 2-term strategies are assumed to remain the same for 3-term problems. The strategies for solving 2-term problems are the same as those used for adding two numbers in 3-term problems.

The rules for modelling CAF, CAL and COF strategies are similar to that of COL above, except for the respective functions that carry out the addition. The functions for CAF and CAL addition are the same, except for their efficiency outputs. They iterate from 1 to the sum of the two addends. The function for COF addition iterates from the first addend to the sum of the two addends, the second addend times.

The following rules model the 'copy the answer to the previous problem' strategy:

(number =I1 =X) (number =I2 =Y) (col =I1 =Y =I2 =X)
---> (copy-answer =I1 =Y =I2 =X)

(copy-answer =I1 =Y =I2 =X) (old-number 0 =ANS)
---> (answer =ANS)

Given a problem, X + Y, and having solved Y + X, the model copies the answer to Y + X, which is represented as *'old-number'* in the working memory. Hence, if 3 + 8 has already been solved, then solving 8 + 3 using this model, produces the following output:

copy-answer 1 3 2 8

4.4.2 Efficiencies

Of the observed strategies, some are more efficient than others. One of the aims of an ITS is to help children make choices between the more and less efficient strategies. To be able to do this, the system needs to know which strategy is more efficient for a given problem. Thus, the production-system modelling includes estimates of the efficiencies of the different strategies. For calculating the efficiencies, it is assumed that for counting, children have a mental representation, for example, they visualize the addends with that number of fingers. The efficiencies are based on the amount of work involved in each strategy, and the demand on a child's memory. The amount of work depends on the number of counts, which depends on the size of the addends. The demand on memory is the effort required to keep a record of the numbers that have been counted and that are left to be counted. The lower the efficiency value, the more efficient the strategy. For the problem, A + B, the amount of work involved in solving it using CAF strategy is A + B (since all the numbers are counted, starting from 1). The demand on memory is a fraction (for example 1/10) of B. It is a fraction because it is assumed that more work is involved in representing the addend and counting it than remembering the number of counts. It is a fraction of B because B is the addend that is counted second and as the counting proceeds, one needs to keep a record of how many of B are counted. Hence, for CAF, the efficiency value is A + B + a fraction of B. For CAL, the amount of work is the same as that of CAF, and the memory demand is a fraction (same as that for CAF) of the smaller one of the two numbers, A or B. For COF, the efficiency is the sum of the second addend, which is the number of counts and a fraction of it. For the COL strategy, it is the smaller one of A and B + a fraction of it.

The 'copy the answer to the previous problem' strategy is the most efficient (but can only be applied in special cases). The efficiency values are meant only for comparing which strategy is more efficient than which, and not by how much. The 'copy the answer to the previous problem' strategy is given '0', the minimum possible, as its efficiency value. Using these criteria for determining efficiencies, the following table shows the relative efficiencies for the example problem, 7 + 12.

Table 4-1. Efficiencies for 7 + 12

copy answer	0
COL	7.7
COF	13.2
CAL	19.7
CAF	20.2

4.5 3-term problems

4.5.1 Strategies

From the empirical work, children's strategies for solving 3-term problems, based on the order in which the additions were carried out, can be categorised into either a *grouping* or a *linear* strategy (described in the previous chapter). Each of these two general strategies includes other more specific strategies. For example, the linear strategy includes 'counting all' as well as 'counting on'. With the grouping strategy, once the two numbers

to be added first are selected, the sum of these two numbers is recalled from memory if it is known as a number fact. The simulation of this part of the strategy is done by looking up a table which represents children's number facts. If the sum is known as a number fact, then this sum has to be added on to the third addend. This is done by using one of the 2-term strategies described in section 4.4 above. If the sum of these two addends is not known, then the problem is two 2-term additions. For our purposes, we only note whether a child solves the problem in the order in which it is written (i.e. linear strategy), or s/he ignores the order and uses the grouping strategy.

The following rule template models the linear strategy, where 'do addition' is one of the 2-term strategies:

(number =A =X) (number =B =Y) (not (used =A =X))
(not (used =B =Y)) ---> (do addition)

The rule instantiates to the first two numbers in the given problem, and the action part adds the two numbers giving another number as the result. The above rule fires again, this time instantiating to the resulting number and the third number in the problem, resulting in the solution to the problem. For example, for the problem 3 + 4 + 7, the rule fires on *number 1 3* and *number 2 4*. The action of carrying out the addition is done by one of the 2-term strategies. Most of the children who used the linear strategy used either the CAF or the COF strategy. Some used the CAL or the COL strategy. After performing the 2-term addition, the result, 7 is added to the working memory, with an arbitrary index, 0 attached to it. The state of the working memory at this stage is:

((number 3 7) (number 0 7) (used 2 4) (used 1 3) (number 2 4) (number 1 3))

The above rule fires again, on *number 0 7* and *number 3 7*. The addition is performed using the same 2-term strategy as that used on the first two

addends, resulting in *number 14* as the solution to the problem. This leaves the state of the working memory as follows:

((number 0 14) (used 3 7) (used 0 7) (number 3 7)

(number 0 7) (used 2 4) (used 1 3) (number 2 4) (number 1 3))

For the grouping strategy, the left-hand side of the above rule requires three numbers, and the 'do addition' part involves trying out the different combinations of two numbers to find out if their sum is already known. The trace in Figure 4-1 shows the actions of a sequence of rules that 'fire' for solving the problem 9 + 6 + 4, using the grouping strategy. To decide which two of the three numbers to add first, the model tries all the three combinations (9 + 6, 6 + 4 and 9 + 4), looking up a working memory of known facts each time to find out if it knows the sum of the two numbers, until it succeeds. The second line of the trace, 'lookup unsuccess 9 6' means that the program does not have 9 + 6 as a prestored known fact. The result of a successful 'lookup' is 'part-answer' of the two addends and the 'left-over' addend. At step 4, 6 + 4 = 10 is known as a number fact. At this step, the interpreter records that 6 and 4 have been used. Step 5 tells us that there is a number left over (not used). Step 6 is the result of a rule which combines a 'part-answer' and a 'left-over' addend to make a 2-term problem. The two numbers are then added using the 'count-on from the larger addend' strategy to get the final answer. The 9 is used at this step. At step 7, the program halts, since the three addends have been used and there is a resulting number (19).

1) Group-first 9 6 4

2) Lookup unsuccess 9 6

3) Group-second 9 6 4

4) Lookup success 9 10

5) Left-over 9, Part-answer 10

6) Count-on-from-larger 10 9

7) Answer 19

Figure 4-1. Trace of 'actions' for grouping strategy on 9 + 6 + 4

4.5.2 Efficiencies

Determining efficiencies of the strategies for solving 3-term problems is dependent on the problems. One strategy might be more efficient than another for a particular problem, whereas the opposite might be true for another problem. For example, for problems like 1 + 1 + 1 and 2 + 1 + 1, the linear strategy might be more efficient, depending on the 2-term strategy used. For most problems, the grouping strategy is more efficient than the linear strategy. For the linear strategy, the efficiencies are calculated in a similar way to that for 2-term strategies. The grouping strategy requires grouping two of the three addends, for which the sum is either known (as a number fact), or is easier/faster to calculate. The amount of effort needed to group two numbers, and to lookup the known sum, is a fraction of that required to calculate the sum. Ignoring problems with more than one '1', for which the linear strategy might be more efficient, each grouping is given an arbitrary efficiency value such that it is always lower than that for computing the sum. The assumption is that 'looking up' to find out if the sum of two of the addends is known as a number fact is more efficient than computing their sum. Each grouping in our case is given the efficiency value 0.5. The efficiencies of the two strategies for the problem, 1 + 4 + 6, is illustrated below:

linear strategy	6.6	(1.1 + 5.5)
grouping strategy	1.6	(1.1 + 0.5)

Note that in both strategies, we assume that the strategy for adding two numbers is COL, the most efficient of CAF, CAL, COF, COL.

4.6 Matching models to data

The models of children's strategies in PALM were constructed from the empirical data. They can be related to the data for all the subjects since the rules are *general* enough to include every child's strategy. The focus in this thesis has been on strategies related to commutativity and associativity, and hence the strategies are modelled at the level of the addend order. Detailed variations of the general strategies have not been implemented.

For three term problems, there are two general models (grouping and linear strategy, Table 4-2) that account for every subject's strategy. A larger set of varied strategies exist for performing the additions, for example partitioning, rounding up and use of number facts such as doubles. In the empirical studies, for those children who used other, more specific strategies that PALM has not modelled, it was noted whether they ignored addend order or considered the addends from left to right. PALM's model for the general grouping strategy accounts for those strategies that ignore addend order. Similar specific strategies that consider the addends from left to right are accounted for by the linear strategy.

The models for 2-term strategies include CAL, COL, CAF, COF and 'copy answer'. Since approximately as many students used 'counting on' strategies (COF, COL) as those who used 'counting all' strategies (CAF, CAL), the empirical studies recorded this detail as well. Informal methods were noted for the addend that was considered first. The children who used informal methods and started from the first addend were classed as using COF. Those who started from the larger addend were classed as using COL. Hence, the models of the two counting on strategies can be further divided into counting strategies and informal methods.

Table 4-2. Summary of models

Strategy	Models
CAF	(number i1 x) (number i2 y) (adjacent x y) (not (used i1 x)) (not (used i2 y)) ---> (caf i1 x i2 y) (caf i1 x i2 y)) ---> (fn cafadd i1 x i2 y) (used i1 x) (used i2 y)
CAL	(number i1 x) (number i2 y) (adjacent x y) (not (used i1 x)) (not (used i2 y)) ---> (cal i1 x i2 y) (cal i1 x i2 y)) ---> (fn caladd i1 x i2 y) (used i1 x) (used i2 y)
COF	(number i1 x) (number i2 y) (adjacent x y) (not (used i1 x)) (not (used i2 y)) ---> (cof i1 x i2 y) (cof i1 x i2 y)) ---> (fn cofadd i1 x i2 y) (used i1 x) (used i2 y)
COL	(number i1 x) (number i2 y) (adjacent x y) (not (used i1 x)) (not (used i2 y)) ---> (col i1 x i2 y) (col i1 x i2 y)) ---> (fn coladd i1 x i2 y) (used i1 x) (used i2 y)
Linear	(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y)) ---> (add i1 x i2 y) ... (applied twice to solve a 3-term problem; add can be any one of the above 2-term strategies)
Grouping	(answer z) --> (stop) (a + b + c) --> (group-first a b c) (group-first a b c) --> (fn lookup-first a b c) (lookup success x y) --> (part-answer x) (left-over y) (part-answer x) (left-over y) --> (fn coladd x y) (lookup-first unsuccess a b c) --> (group-second a b c) (group-second a b c) --> (fn lookup-second a b c) (lookup-second unsuccess a b c) --> (group-third a b c) (group-third a b c) --> (fn lookup-third a b c) (addd a b) --> (fn addd a b) (lookup-third unsuccess a b c) --> (do-it-anyway a b c) (do-it-anyway a b c) --> (fn do-it a b c)

In the following sections, we relate the production-rule models summarised in Table 4-2 to the data.

4.6.1 Snapshot data

Table 4-3 shows the number of children in the pilot study (n = 21) whose strategies could be matched by the 2-term models in Table 4-2.

Table 4-3. Number of children in the pilot study described by each model

model	no. of children
CAF	4
COF	1
COL	14
CAL & CAF	1
COL & CAF	1

The model for the COL strategy would account for two-thirds of the data. The proportion of children who are accounted for by each model depends on the subjects. In the pilot study, all the children were over six years of age, and most of them were using advanced and efficient strategies. If the sample consisted of children younger than five, then the dominant model would be CAF. If the sample consisted of children who were all at approximately the same performance level, then it is possible that fewer sets of models would be required to model their strategies.

The last 2 children's performance (Table 4-3) would be accounted for by two models. The model that describes his/her performance depends on the strategy s/he uses on a given problem. This would be fine for a system that models a child's strategy on every problem. An ITS in this domain will have to model on every problem in order to achieve a detailed description like "This child is using COL sometimes and CAF at other times".

Table 4-4 presents a summary of the detailed analysis of the 3-term strategies that were observed in the main study (n = 77) and the number of children in each category. It also shows the models in PALM that represent the categories. The categories are as follows:

1 - grouping

2 - explicit knowledge of grouping but did not use it

3 - applied commutativity to first 2 terms

4 - no evidence of transfer of commutativity

5 - did not know commutativity but used the COL strategy

6 - linear strategy and did not know commutativity

The table is derived from the results in Appendix 4 (see also Figure 3-1, chapter 3).

Table 4-4. 3-term strategies in the main study

category	no. of children	model
1	50	grouping
2	2	linear
3	7	linear
4	3	linear
5	1	linear
6	14	linear

The model for the grouping strategy (Table 4-2) would account for all the children (n = 50) who used that strategy (category 1, Table 4-4). The other children (n = 27) would be best described by the linear model. As can be seen from Table 4-2, the production-rule models in PALM do not make a distinction between categories 2, 3, 4, 5 and 6. With the current implementation, PALM could have a separate model for category 3 that would distinguish students who applied commutativity to the first two terms of a 3-term problem from those who used the grouping or the linear strategy. However, PALM would not be able to model categories 2 and 5, since it does not have a means for representing conceptual knowledge. This remains a challenging extension to PALM (For an ITS, it is possible to diagnose categories 2 and 5 by interrogating the child). Categories 4 and 6 would be distinguished by incorporating descriptions of children's performance on 2-term problems. Categories 2, 4 and 6 (70% of those

children modelled as linear) would be described by the linear strategy, ignoring distinctions between their conceptual knowledge.

In sum, the models in PALM match 'procedural data' well. Its limitation is distinguishing the conceptual knowledge associated with the procedures.

4.6.2 Longitudinal data

Referring to the results of the development of strategy (Table 3-5) for solving 2-term problems, there is a definite trend in transition to more efficient strategies. The transitions observed in the 12 subjects over the 20-month period can be summarised as follows:

CAF --> COL

CAF --> CAF & COL --> COL

CAF & COF --> COF --> COL

CAF --> COF --> CAL & COF

CAF --> CAL --> COL (2 children)

CAF --> CAF & COF

CAL --> COL

The other 4 students had used the same strategy throughout, i.e. there was no observed strategy transition. Note that the time period between the strategy transitions varied from one individual to the next. Details of time intervals at which the transitions took place is presented in Table 3-5. To summarise the data for the development of strategies, the transition is from CAF as the most basic and inefficient strategy to COL as the most efficient one. Intermediate between these are CAL and COF. It seems that every child would get to the most efficient strategy, but it is not certain whether

everyone goes through all the intermediate steps. For example, the child who went through the following sequence of strategies

CAF --> CAF & COL --> COL

does not seem to have gone through CAL and COF. There is a possibility though, that she did go through them, but did not use them at the interview. The detailed sequence in which each child's strategy transition took place varies from another. There is only one sequence which was the same for two children. Even for them, the time intervals after which the transitions were observed were different.

The models in Table 4-2 would be able to describe each child's strategy on any given problem. As in the case of the data for the pilot study, when modelled over a set of problems, there would be more than one model that describes the performance of those children who used more than one strategy at a particular interview session.

PALM's production-rule models are not capable of modelling the transition of strategies over time. The objective of the learning mechanism described in the next chapter is to model the transition from one performance level to another.

4.7 Discussion

With respect to the *order* in which a given 2- or 3-term problem is solved, the models are capable of accounting for any individual's strategy. However, more distinctions can be made between their strategies if details like conceptual knowledge and the use of number facts are also modelled.

The models can be used for diagnosing a child's strategy at a particular time. Although there are definite trends in the development of strategy, for

example from CAF to COL, PALM's production-rule modelling component cannot model them. Models of learning are needed to explain such trends.

The efficiency estimates work very well for ordering the strategies that are modelled, in terms of their relative efficiencies. However, this method of estimating efficiencies may need revising when the other strategies are modelled as well. It would certainly need revision in the strategies for solving problems with operators like multiplication.

The internal details of the strategies, such as the one by one counting of an addend, have not been implemented using rules. If this were done, then the efficiencies of the strategies could be modelled using rule firings. This would provide better justifications for arriving at efficiency values. Furthermore, the simulation of efficiencies of strategies using traces of rule firings would be able to take into account other strategies, for instance informal methods, that have not been implemented in PALM. Such an assessment of efficiencies could also be used for problem solving strategies involving the other arithmetic operators.

Note that the production-rule models represent procedural knowledge only; if there is any conceptual knowledge, it is assumed in the procedural knowledge. Evidence from the empirical work reported in chapter 3 and from other research (Hennessy, 1986; Resnick, 1983) strongly suggests the need to make a distinction between these two types of knowledge. Self (1988) highlights the need for student models to include descriptions of conceptual knowledge in addition to purely procedural knowledge. For example, the production-rule models of performance on 2-term problems tell us whether a child copies an answer to a previous commuted problem or not. This indicates that such children know and use the concept of commutativity, but for those who do not use the concept, it does not tell us whether they know it or not. This is one example where diagnosing a child's conceptual knowledge is important. One possible way to diagnose conceptual knowledge is by modelling tasks that were presented to the students in the

empirical work, and the measures that were used to categorise the different levels of the concept of commutativity (chapter 3). One such task was a pair of subtraction problems, e.g. 5 - 3 and 3 - 5. If a child generalizes the concept of commutativity to this problem, then it shows that s/he has some understanding of the concept.

4.8 Summary

This chapter described some possible architectures for production systems. Examples of existing production systems have been presented. The components of PALM were introduced. Its interpreter, the process of matching and conflict resolution strategies were described. The description language for representing the input problems, the rules and items in the working memory of the production system have been outlined. The empirically identified strategies for solving addition problems have been simulated using production rules. This also includes estimates representing efficiencies of the strategies. The models are descriptions of children's strategies at different levels of development. This does not explain where the strategies in such models come from and how they got there.

In the following chapter, an attempt to model the transition process from one performance level to another, is discussed. The model begins with rules representing the ability to solve 2-term problems (e.g. 3 + 4) as prerequisite knowledge for learning. When presented with a 3-term problem like 3 + 4 + 5, which is of a type that has not been encountered before, PALM's current rules are not adequate to solve it. As a result of this kind of failure, PALM learns to solve the new type of problem.

Chapter 5

MODELLING LEARNING

5.1 Introduction

The last chapter was an account of the development of production-rule models that describe children's strategies for solving arithmetic problems related to the concepts of commutativity and associativity. The models describe 'snapshots' of children's performance, that is performance at one particular time only; they do not explain change in performance over time. A model of learning or a model for transition from one snapshot to the next, in addition to the production-rule models of the snapshots, would provide a more complete model of children's behaviour. This chapter describes the design and implementation of a model of learning for the transition from one of the performance levels of arithmetic problem solving to another. The computational model of the acquisition of procedural knowledge of associativity uses knowledge of commutativity (procedural, at least) as prerequisite knowledge. The model begins with rules representing the ability to solve 2-term problems. When presented with a 3-term problem, which is of a type that has not been encountered before, none of the current rules are applicable. In this situation, PALM learns as a result of failure. This is done by generalizing its existing rules. Once it has learnt to solve 3-term problems, it is driven by the goal of learning more efficient strategies.

The learning part of PALM consists of 2 components:

i) Failure-driven learning, which occurs when there are no applicable rules to solve a given problem. The program learns by generalization. For example, it solves 3-term problems like 4 + 5 + 3 by generalizing its rules for solving 2-term problems like 3 + 4.

ii) Efficiency-driven learning, an ACM-like (Langley, Ohlsson and Sage, 1984) learning component. It learns features of problems as conditions for which two of the three numbers to add first, in order to solve the problem most efficiently.

Failure-driven learning is based on the assumptions that experience and previous knowledge facilitate learning and that learning takes place when one is 'stuck'. Efficiency-driven learning assumes that learning is driven by a search for a low effort solution.

The chapter is organized as follows. The first section of the chapter provides an introduction to some of the types of machine learning techniques. This is followed by an outline of the learning components of PALM and a discussion on the motivation for the particular learning methods it employs. The next section presents the computational details of the learning mechanisms. The final section is a discussion of the learning mechanisms and also includes suggestions for extensions of the implementation.

5.2 Machine learning techniques

As outlined above, PALM includes learning by generalization (failure-driven learning) and by specialization (ACM-like operator applicability). This section introduces three main types of learning that have been addressed in the machine learning literature - learning from examples, learning by analogy and explanation-based learning. These three types of learning do not, by any means, cover all approaches to machine learning, but do illustrate a representative sample. In previous machine learning systems, the types of learning mechanisms that are in PALM are normally incorporated within the method of learning from examples. Hence, it is this type of learning that the discussion below covers in more detail.

5.2.1 Learning from examples

Learning from examples is the method of arriving at a rule, hypothesis or description of a concept from a set of positive and negative examples. Despite its apparent simplicity, the approaches to learning from examples are nearly as numerous as the people who have worked on it (Langley and Carbonell, 1984). Regarding the examples, some of the ways in which systems that learn from examples differ are the type of examples, the way they are presented and the number of examples. There are two classifications according to the types of examples:

- systems that learn from positive examples only
- systems that learn from a set of positive and negative examples.

Along the dimension of the way the examples are presented, learning from examples can be:

- one-trial
- incremental

In the one-trial method, learning is based on all the examples considered at once. Some learning from examples systems that employ the one-trial method are Michalski's Star methodology (Michalski, 1983), ID3 (Quinlan, 1983) and ACM (Langley, Ohlsson and Sage, 1984). The incremental method is where the examples are considered one at a time, and at each stage, the previous learned rule or hypothesis is refined accordingly. Examples of systems that employ this method are Winston's ARCH program (Winston, 1975), SPROUTER (Hayes-Roth and McDermott, 1977), Thoth (Vere, 1977) and the version spaces (Mitchell, 1977, 1978, 1982). Incremental learning methods provide much more plausible accounts of the human learning process (Langley and Carbonell, 1984; Langley, Ohlsson and Sage, 1984).

The third, minor distinction regarding examples is the number of examples. This varies along several dimensions; for example, the objective of the learning system, its application, the precision of the learning outcome, the availability of examples and the task domain. A special case of learning from examples is explanation-based learning (described in section 5.2.2 below), which requires only a single example.

Furthermore, within the learning from examples approach, systems can differ in their detailed mechanisms for learning. There are two main mechanisms for learning: discrimination and generalization.

Discrimination

Discrimination (or specialization) is a mechanism for learning that involves the creation of a new rule, or modification of an existing one, so that it is less general than an existing rule, while still retaining the same actions. When using a set of positive and negative instances to arrive at a description that covers the positive instances, learning by specialization is usually used to make the description specific so that it does not include the negative examples. Discrimination-based learning programs normally use discrimination networks for representing knowledge. Some examples of such systems are ID3 (Quinlan, 1983), which is a descendant of CLS (Hunt et al., 1966) and ACM (Langley, Ohlsson and Sage, 1984). Figure 5-1 shows a discrimination network representing the concept **black or (not black and circle)**. '+' means that all the instances at that node are positive and '-' means all the instances are negative.

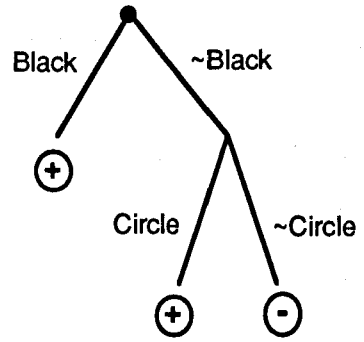


Figure 5-1. A sample discrimination network

The discrimination network representation is used to arrive at a hypothesis that discriminates the positive instances from the negative ones in a given set of instances. Discrimination begins at the top node of a network, and proceeds with one branch at a time. In the example above, the system would begin with the **black or not black** branch. Next, it would create a branch coming from one of the new nodes, if necessary. The tree grows downward until terminal nodes are reached which have either all positive or all negative instances. The path(s) leading to the 'all positive' nodes form the description of the concept.

Generalization

Generalization is a mechanism of learning which involves creating a new rule, or modifying an existing one, so that it is more general than an existing one, while the actions remain the same. The term generalization is also used for the process of arriving at a rule or hypothesis that describes a set of instances of a concept. Positive and negative instances are used in a different manner from the way they are used in discrimination-based learning. Rather than looking for differences between positive and negative instances, generalization-based learning looks for features held in common by all positive instances. Examples of learning from examples systems that learn by generalization include SPROUTER (Hayes-Roth and McDermott, 1977), Thoth (Vere, 1977) and Winston's program for learning structural concepts such as 'arch' (Winston, 1975).

There are several ways of implementing generalization. Some of these are:

i) *Dropping condition rule* - to generalize a conjunction, drop any of its conjunctive conditions. For example, given the current description of a class of objects, **black, large and circle** and a new instance, **white, large and circle**, the description can be revised by dropping the condition for the colour of the objects. Hence, the generalized description, **large and circle** covers the new instance.

ii) *Turning constants to variables*. In the above example, the concept description could be generalized by turning the constant **black** to a variable:

?colour, large and circle

iii) *Adding disjunction rule* - to generalize a conjunction, change it to a disjunction. To include the new instance in the above concept description, it could be generalized by adding a disjunction:

(black, large and circle) OR (white, large and circle).

iv) *Climbing generalization tree rule*. For example, using the generalization tree in Figure 5-2, an instance including **triangle** and another including **rectangle** in their descriptions can be generalized to **polygons**.

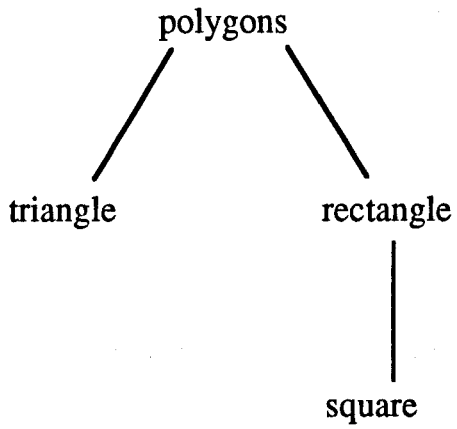


Figure 5-2. A generalization tree

Further and more detailed rules of generalization are described in Dietterich and Michalski (1983) and Michalski (1983).

5.2.2 Explanation-based learning

Explanation-based learning (EBL) is a technique for obtaining generalized concept definitions based on an analysis of one example only, using a set of facts about the domain. The facts include abstract rules of inference about the domain. A high level description of the target concept (*goal concept*) and a definition of what an acceptable concept definition (*operationality criterion*) would be is also provided. EBL works by constructing explanations of why the training example satisfies the goal concept. This is done by expanding the terms in the high level description until all the terms in the description meet the operationality criterion. Then the explanation is generalized to form a rule which is capable of matching instances of the goal concept. In brief, the goal of EBL is to redefine a given concept in operational (usable) terms.

Table 5-1 presents an example for learning the concept cup, borrowed from Mitchell, Keller and Kedar-Cabelli (1986).

Table 5-1. An example of the EBL approach

Given:

- *Goal concept*: The concept of a cup:

  open-vessel(o) & stable(o) & liftable(o) --> cup(o)

- *Training example*:

  part-of(obj1, concavity-1)

  isa(concavity-1, concavity)

  is(concavity-1, pointing-up)

  part-of(obj1, bottom-1)

  is(obj1, light)

  ...

- *Domain theory*:

  is(x, light) & part-of(x, y) & isa(y, handle) --> liftable(x)

  part-of(x, y) & isa(y, bottom) & is(y, flat) --> stable(x)

  part-of(x, y) & isa(y, concavity) & is(y, pointing-up) --> open-vessel(x)

  ...

- *Operationality criterion*: The concept must be defined in terms of predicates used in the example.

Determine: An operational description of the goal concept that covers the training example.

Note that the domain theory contains at least one rule with 'cup' in the right-hand-side, and that it contains rules which mention features from the training example in the left-hand-side. Both are necessary in order to generate an operational definition of the concept.

The basic approach to generalization involves two steps:

i) *Explanation* - the domain theory is used to construct an explanation that proves that the training example is a positive instance of the goal concept. Figure 5-3 presents an explanation for the training example of the concept 'cup' given in Table 5-1. Note that the top node in the explanation tree refers to 'cup', and each of the terminal nodes refer to propositions in the training example. If a proposition was missing in the training example, then the explanation of the concept would not be complete. For example, if part-of(x, y), was missing, then the description stable(obj1) would not have been achieved, and hence the goal concept would not be able to be explained.

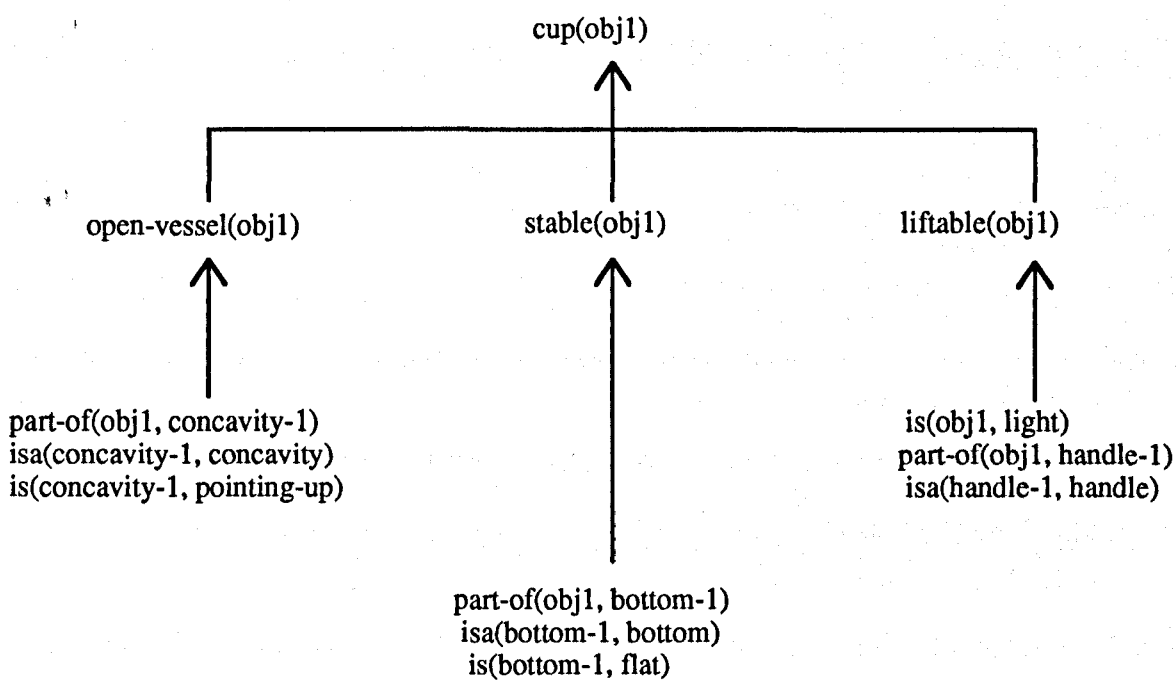


Figure 5-3. Explanation for an instance of 'cup'

ii) *Generalization* - the terminal nodes are transformed into a set of conditions under which the explanation will hold. The most general version of the explanation in step i) which is consistent with the domain theory is achieved using generalization rules like replacing constants with variables.

The operational definition of the concept in Table 5-1 from the explanation in Figure 5-3 is as follows:

part-of(x, xc) & isa(xc, concavity) & is(xc, pointing-up)

& part-of(x, xb) & isa(xb, bottom) & is(xb, flat)

& part-of(x, xh) & isa(xh, handle) & is(x, light) ---> CUP(x).

The EBL approach has several advantages. Firstly, it requires only one example; negative examples are not required at all. Secondly, a justification is provided for the concept description that is generated. Finally, it is capable of handling noisy data, since these will be identified in the explanation process. Provided the domain knowledge is complete, if an explanation cannot be constructed, it would imply that the data (example) is noisy. The approach also has some disadvantages. Firstly, it requires significant domain knowledge, which restricts its application to domains where such knowledge is available. Secondly, as pointed out by DeJong and Mooney (1986), the generalization obtained from one example only can be biased or specific to that particular training example.

> "It often does not generalize the new concept far enough from the particular training example. The result is undergeneralizations that reflect many unimportant details of the example problem." (DeJong and Mooney, p. 146).

Current development of research on learning from examples incorporates EBL in order to benefit from both approaches. Some of the limitations of the

explanation-based approach can be counteracted by the learning from examples approach and vice-versa. Some of the approaches that have been used for combining EBL with learning from examples are:

i) Applying learning from examples to obtain a set of possible generalizations. This set is then pruned and refined using EBL, to obtain an explained/justified (in terms of domain theory) generalization. An example of a system utilizing this approach is UNIMEM (Lebowitz, 1986a, 1986b).

ii) Applying EBL to each example and learning from examples to the resulting generalized examples. Such an approach is employed in the WYL program (Flann and Dietterich, 1986).

5.2.3 Learning by analogy

Learning by analogy is the process of applying existing knowledge to a new domain by recognizing similarities between the two domains, and then finding the transformation that when applied to information in the previous domain, will yield new information that works in the new domain. There are two main ways in which analogy has been applied to problem solving. Figure 5-4 illustrates the first one, called *transformational analogy*, in which the solution to an existing problem is transformed onto an analogical one. The method matches an old problem similar to the new one and transforms the final solution to the old problem to the new one. The match between the two problems is used to guide the transformation process. The procedure for reaching the solution to the old problem, that is, the mapping between the old problem and its solution is ignored; it is a solution to solution mapping. Problem solving using this type of analogy has been explored by Carbonell (1983).

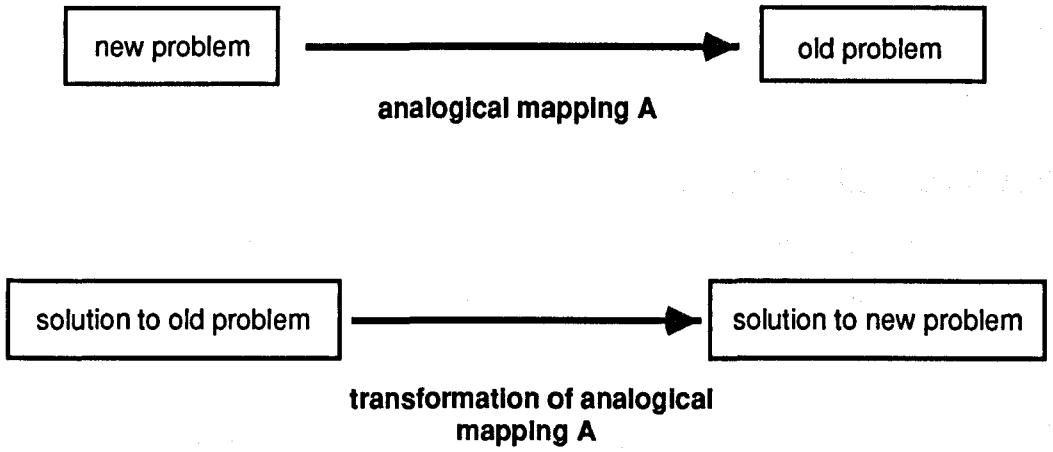


Figure 5-4. The process of transformational analogy

The second method is called *derivational analogy*. The basic idea is illustrated in Figure 5-5. Unlike the previous method which transforms the solution to the existing problem onto the new problem, this method considers two things: an analogy between the two problems, and a *derivation* of the solution of the old problem. It then replays the derivation to solve the new problem. This type of analogy has been examined by Carbonell (1986).

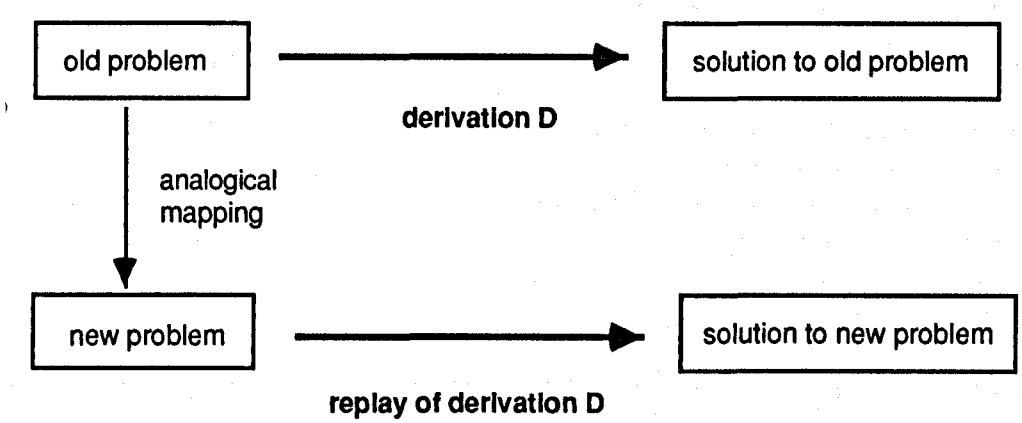


Figure 5-5. The process of derivational analogy

5.3 Learning mechanisms in PALM

In this section, the choice of the learning mechanisms in PALM and the motivation for the choices, based on the empirical work are discussed. An introduction to the learning components of PALM is provided first.

5.3.1 Outline of PALM's learning components

As mentioned in the previous chapter, PALM has two major components. The learning component is an extension of its production-rule modelling component, discussed in chapter 4. PALM's cycle of matching rules to working memory items, conflict resolution strategies, etc. remain the same with or without the learning component being active. The motivations for learning are failure and minimizing the amount of work required, i.e. efficiency. Failure-driven learning occurs when PALM has rules for solving 2-term problems, like 4 + 5, but does not have rules for solving 3-term problems, like 5 + 7 + 6. After learning to solve 3-term problems, PALM learns more efficient strategies, like grouping. Figure 5-6 is an outline of the learning components of PALM. It shows efficiency-driven learning being applied to 3-term problems. Note that it could be applied to 2-term problems as well.

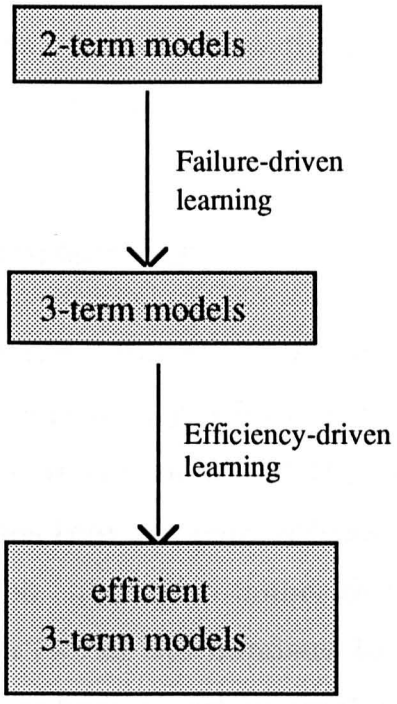


Figure 5-6. The learning components of PALM

Failure-driven learning

Learning in PALM is driven by failure when there are no rules that are applicable to its current problem-solving state. The program tries to adjust its existing knowledge to apply to the new situation. After solving 2-term problems, when faced with a 3-term problem, like 3 + 4 + 5, none of its current rules are applicable. The existing knowledge in this case is the set of rules for solving 2-term problems. An example of such a rule is:

(number a x) (number b y) (adjacent x y) (not (used a x))
(not (used b y)) ---> *(do addition)*

The above rule adds two numbers that are adjacent to each other. It applies to 3-term problems to get only a partial solution. For example, for the problem 5 + 4 + 5 (represented as (number 1 5) (number 2 4) (number 3 5)), the rule applies to 5 + 4. The next step is to add the resulting 9 and the remaining 5. PALM is not able to do this because the two numbers are not

adjacent to each other. As a result of this failure, PALM learns to complete the solution to such 3-term problems by generalizing its existing rules.

Efficiency-driven learning

As a result of the failure-driven learning, PALM is able to add numbers that are not necessarily adjacent to each other. For example, for the problem above, it can now add the two 5's together first. The program learns strategies that are more efficient than those it already knows. To do this, it draws on children's number facts. The goal is to apply number facts to select which two of the numbers in a 3-term problem to group and add first. The system begins with general rules representing basic strategies for solving 3-term addition problems, and a set of operators, like **(equal x y)**, representing children's number facts. It tests whether an operator is applicable to a given problem. If it is, then that operator becomes an additional condition of the initial rule which adds two numbers. Hence the learning problem is one of finding conditions where the number facts are applicable. This is discussed in more detail in section 5.4 under computational details.

5.3.2 Choice of learning mechanisms in PALM

The learning mechanisms in PALM are closest to learning from examples. More specifically, they could be seen as an incremental method of learning from examples, and concentrate on the first example only. In the discussion that follows, some of the reasons for not using analogy and explanation-based learning are highlighted.

The transition being studied here is that from the ability to solve 2-term problems to that of 3-term problems. To solve 3-term problems, children 'split' them into two 2-term problems. This process does not appear to be analogy. It seems more likely that it is generalization. Furthermore, for

analogy to be applied, the problems must be fairly close to each other. All current analogy systems involve a one-to-one correspondence such that each element maps to another. Referring to a similar kind of problem with analogy that is addressed here (matching two problems with different numbers of addends), Keane (1988), argues that most current analogy theories face the difficulty of not being able to match two relations with different numbers of arguments, even though they may be very similar. For example, for the problems, 3 + 5 and 2 + 7 + 4, analogy would require mappings between 3 and 2, 5 and 7, and similarly an item that would correspond with the 4. Since there is no corresponding element that could represent a mapping for 4, computational analogy could not be applied between 2-term and 3-term problems.

The learning mechanisms in PALM share two features in common with explanation-based learning. First, they both use only one positive example. Second, both EBL and the mechanisms in PALM use domain knowledge. Note that domain knowledge in PALM serves a slightly different purpose to its role in EBL. In failure-driven learning, domain knowledge is knowledge of ability to solve 2-term problems, which is used to solve 3-term problems. In efficiency-driven learning, domain knowledge is existing knowledge which is used to learn more efficient strategies. Explanation-based learning proper is not used here because our aim is to solve a new type of problem, and not to generalize the description of the example problem.

Failure-driven learning : generalization

Children who use the linear strategy for solving 3-term problems generalize from their previous knowledge of solving 2-term problems to add the first two numbers. Then they write the sum of the first two addends and the third addend as another 2-term problem. For example,

4 + 7 + 5 =

4 + 7 = 11

11 + 5 = 16

4 + 7 = 11 is a generalization of 2-term addition. The transition from the 'count on from larger addend' (COL) strategy on 2-term problems to the grouping strategy on 3-term problems provides more evidence of generalization. Ignoring addend order on 2-term problems leads to ignoring addend order on 3-term problems. For example,

3 + 8: 9, 10, 11

5 + 7 + 5: 5 + 5 = 10: 10 + 7 = 17

Further evidence for generalization from the empirical work reported in chapter 3 is provided by subtraction problems where the minuend is smaller than the subtrahend (e.g. 3 - 5). Performance on such problems revealed children's generalization of their existing problem solving knowledge to solve a new type of problem. This is strong evidence for the possibility of generalization from 2-term to 3-term problem solving.

Furthermore, there is ample evidence from previous research of children using generalization in their problem solving. For example, in subtraction, there is evidence of students who always borrow after having consistently seen examples requiring borrowing (VanLehn, 1987). Matz (1982) and Sleeman (1984) provide evidence of generalization in the domain of algebra.

PALM's generalization mechanism considers learning from a single example. This choice has been influenced by previous research which suggests the validity of generalization from a single example (Ahn, Mooney, Brewer and DeJong, 1987; Anderson, 1989; Elio and Anderson, 1981).

Finally, generalization has been chosen, as opposed to discrimination, because only a positive example is considered. Negative examples are not considered and hence there is no need to discriminate between positive and negative examples.

Efficiency-driven learning : condition learning

The choice of the condition learning mechanism in PALM has been influenced by condition learning in ACM (reviewed in chapter 2), and by Langley's (1987) theory of discrimination learning. Both ACM and Langley's theory use the production-system formalism, where learning is modelled by creating new rules. A set of general rules representing an initial problem solving state and a set of conditions for generating a new state are provided. The learning problem is then one of finding the conditions, and to which problem state they should be applied. Conditions are learnt in order to form new rules which are more *discriminant* than the initial set. A rule is a discriminant version of another when the first rule has conditions that are special cases of the second, and the action sides of the two rules are the same.

The second motivation for choosing the learning mechanism in efficiency-driven learning is the use of existing knowledge (i.e. number facts) to reach a more advanced stage in problem solving. This existing knowledge is represented as the set of conditions that is provided to the learning system.

As children gain more experience in solving arithmetic problems, their strategies become more efficient. Furthermore, as the strategies become more efficient, they include more number facts. In the empirical studies reported in chapter 3, there was substantial evidence of children using strategies based on number facts. For example, for the problem, 4 + 5 + 4, performing 4 + 4 first was a common strategy. When asked why they had chosen to do the 4 + 4 first, a common reply was "because I know 4 and 4".

On a problem like 3 + 6 + 4, children who chose to do 6 + 4 first explained "because I know that 6 + 4 = 10" or "because that makes 10" or "because it is easier to deal with a 10", and so on. Further empirical evidence of children using number facts in their strategies will be quoted in the following sections.

5.4 Computational Details

The basic outline of the learning mechanisms in PALM was presented in section 5.3.1 above. This section provides details of the two learning mechanisms.

5.4.1 Failure-driven learning

The implementation of sets of production rules that represent strategies for solving problems like 3 + 5 and 6 + 4 + 7 in PALM was described in chapter 4. Such problems are given as input to the production system, and the interpreter selects and executes the appropriate rules for solving them. It is able to solve those types of problems for which it has explicit problem-solving rules. However, when a different type of problem is encountered, it is not able to solve it because it does not have rules for solving this particular type of problem. The way PALM solves such closely related, new types of problems is discussed in this section.

Failure-driven learning in PALM takes place when PALM does not complete solving a given problem. Completion of the problem solving is determined by reaching the endstate, that is arriving at an answer to the problem. When PALM has not successfully completed solving a problem, and it does not have any more rules that match the current problem solving state, it learns by restructuring its existing rules. It generalizes its rules to apply to the current problem. We shall assume that PALM is capable of solving 2-term problems only, and not 3-term problems. Hence, PALM does

not contain rules for solving 3-term problems. The following set of rules is a model for one of the strategies for solving 2-term problems:

(number i1 x) (number i2 y) (adjacent x y) (not (used i1 x)) (not (used i2 y))
---> (col i1 x i2 y)

(col i1 x i2 y) ---> (fn coladd i1 x i2 y)

(addd i1 x i2 y) ---> (fn addd x y) (used i1 x) (used i2 y)

Given a 3-term problem, PALM applies its 2-term rules in an attempt to solve it. The following is a trace of the actions after the above rules are executed for solving 7 + 15 + 6. The function *SSTART* starts the execution of the program.

(SSTART '((number 1 7) (number 2 15) (number 3 6)))

1) (adjacent 7 15)

2) (col 1 7 2 15)

3) (addd 2 15 1 7)

4) (number 0 22) (used 2 15) (used 1 7)

The first step is a result of the knowledge of adjacency:

(number 1 x) (number 2 y) ---> (adjacent x y)

The program stops after the 2-term rules solve 7 + 15. As discussed in the last chapter, there are two reasons for the interpreter coming to a 'halt'. Firstly, if the problem has been solved, and secondly if there are no more applicable rules. In the example above, the program stops because of the latter. The working memory at this state is as follows:

((number 0 22) (used 2 15) (used 1 7) (addd 2 15 1 7) (col 1 7 2 15) (adjacent 7 15) (number 1 7) (number 2 15) (number 3 6))

In order to complete solving the problem, (number 3 6) and (number 0 22) need to be added. The first rule above should match with these two numbers, but it cannot since 22 and 6 are not adjacent to each other. PALM generalizes its rules by dropping a condition from one of its rules. The *dropping condition* generalization rule has been chosen because of its simplicity, and because of its applicability to the particular transition that is studied here.

In order to choose a condition to drop, PALM assesses the conditions of its rules to find those that probably caused the premature halt of the system. This is done by evaluating whether the conditions of rules match the current state or not. The preferred conditions to drop are those that do not match, since it is likely that if these conditions had matched, then the rule containing the conditions could have fired. In the current implementation, only one such condition in a rule can be dropped. A condition can only be dropped if it is the only one in the selected rule that does not match. This is because when there are several possible conditions to drop, there is no principled way of choosing which one(s) to drop. Furthermore, in such a case, all the conditions that do not match will need to be dropped for the rule to be able to 'fire'.

Another reason for dropping only one condition is that the aim of the program is to create as little change in the original rule as possible, so that it applies to the problems it could solve previously as well as to the new type of problem that it is trying to solve. Since it is learning by generalization, and since the new problem is quite similar to the previous problems, minimum change to the original rule is desired.

If there is more than one rule whose condition can be dropped, then PALM chooses the first rule. The first rule is chosen because the rules in the

ruleset are ordered. Rule ordering is one of the conflict resolution strategies in PALM (see chapter 4). The firing of the later rules usually depend on the earlier ones. Hence, if the earlier ones do not fire, then it is probable that this is the reason for the later rules not having items in the working memory to match to.

Once a condition is dropped and the rule is amended, the interpreter continues its cycle of matching, selecting and executing its rules. If the changed rule becomes applicable then the problem solving continues from the point where the program had 'halted'. If it is not applicable, then the changed rule is replaced by the original one, and the next rule to generalize is chosen until an amendment of a rule leads to all its conditions being satisfied. If this does not succeed, then PALM is not able to complete solving the given problem.

In the above example, the process of finding out whether the conditions of the current set of rules have matching working memory items or not returns:

((yes yes no yes yes) (no) (no))

The three lists correspond to the three rules representing a 2-term strategy:

(number i1 x) (number i2 y) (adjacent x y) (not (used i1 x)) (not (used i2 y))
---> (col i1 x i2 y)

(col i1 x i2 y) ---> (fn coladd i1 x i2 y)

(addd i1 x i2 y) ---> (fn addd x y) (used i1 x) (used i2 y)

The lists show that the third condition in the first rule, and the only conditions in the other two rules do not match. The first rule that has a 'no' is the rule that is selected to be amended. The 'no-condition' of this rule is deleted and the interpreter then checks if it is applicable. By dropping the

third condition in the first rule (i.e. the adjacency condition), the rule becomes applicable. The set of rules is reset with the first rule amended to

(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y))
---> (col i1 x i2 y).

The interpreter continues with the cycle of matching, selecting and 'firing' rules, producing the following output:

(col 0 22 3 6)

(addd 0 22 3 6)

(number 0 28) (used 0 22) (used 3 6)

The interpreter at this stage works out that the problem has been solved, and hence comes to a halt. The final number, 28, is the solution to the problem.

5.4.2 Efficiency-driven learning

After the failure-driven learning, the system is able to solve 3-term problems. Since the numbers do not have to be adjacent to each other, the three numbers can be added in any order. In order to use an efficient strategy, PALM learns specific conditions to choose which two of the three numbers to add first. To begin with, there is a set of initial problem-solving rules which represent the linear strategy, a set of operators that represent children's known facts and a given 3-term problem. The set of operators are derived from the empirical studies of children's arithmetic problem solving. The goal of the efficiency-driven learning is to combine the initial problem solving knowledge and the known facts to achieve a strategy that is more efficient than the initial ones. The following is an example of a rule representing an initial strategy:

(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y))

---> *(fn col i1 x i2 y).*

The condition sides for all the initial problem solving strategies are the same. The action sides include the 2-term addition strategies, COL (rule above), COF, CAL, CAF and 'lookup answer'. Two such 2-term strategies are applied to solve a 3-term problem using the linear strategy.

Presented below are some examples of operators representing number facts that children use for 'shortcut' in their strategies. They are derived from children's grouping strategy for solving 3-term problems.

1. **(equal x y)** - two of the three numbers are the same. For example, for a problem like 6 + 9 + 6, children were often observed to do 6 + 6 first. Their explanations for finding the sum of two equal numbers first were of the type "because I know that 6 and 6 are 12" and "because I already know that one, I do not have to count".

2. **(equal (x + y) 10)** - two of the numbers add to 10. For example, a problem like 3 + 4 + 6 using grouping strategy was often solved by doing 4 + 6 first. Explanations of this choice included "because I know that it makes 10", "because it is easier to add to 10" and "because it makes 10 and it is easier to deal with 10".

3. **(equal (x or y) 1)** - one (or more) of the numbers is '1'. For example, some children solved 4 + 7 + 1 by doing 7 + 1 first. When asked why they had chosen to do that first, the replies included "because it is easier to do that first" and "because it is just 1 number more than that (referring to 7)".

4. **(equal x (y + 1))** - one of the numbers is 1 more than another. For example, Freda solved 7 + 4 + 5 by solving 4 + 5 first:

"I pretend that the 4 was 5

5 + 5 = 10

10 - 1 = 9"

5. **(equal (x + y) z)** - the sum of two of the numbers equals the third. For example, some children who solved 2 + 4 + 6 as 6 + 6, that is by solving 2 + 4 first, gave explanations like "I know that 6 and 6 are 12, ... and that (referring to 2 + 4) gives 6".

PALM learns to apply these operators as conditions for efficient problem solving. Given an input problem, the efficiency-driven learning algorithm works as follows. The operators are tested against the problem to select those that are applicable. The selected operators are added as additional conditions to the left hand sides of the rules for each strategy. The problem is then solved using each strategy, and their efficiencies are compared. The efficiencies are computed in the same way as in chapter 4. The rules representing the most efficient strategy are added to the list of rules, and the inefficient ones are eliminated. If there is more than one equally efficient rule, then all of them are added to the ruleset, and when the problem is solved, one of these rules is chosen. Note that the initial, more general rule is not replaced, since the newly learned rule only applies in specific cases. The operators that have been used are deleted from the list of operators, since rules containing them as conditions are already in the ruleset. When the next problem is presented, PALM repeats the process of finding applicable operators, adding the new rule(s), if any, to the ruleset, deleting the used operator from the list of operators and then solving the problem.

For example, initially, the problem 5 + 7 + 3 would be solved using the linear strategy, that is by adding the 5 and the 7 first. After testing each of the operators, **(equal (x + y) 10)** is the only one that is applicable. It is added as a condition to each of the strategies. The new rule for the 'lookup answer' strategy, for example, is

(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y))
(equal (x + y) 10) ---> *(fn lookup i1 x i2 y).*

The input problem is then solved using all possible combinations of 2-term strategies. Some of the combinations for solving 5 + 7 + 3 are:

i) lookup 7 + 3 followed by lookup 10 + 5

ii) lookup 7 + 3 followed by COL 10 + 5

iii) lookup 7 + 3 followed by CAL 10 + 5

iv) COL 7 + 3 followed by COL 10 + 5

After trying out all combinations, 'lookup 7 + 3 followed by COL 10 + 5' is the most efficient combination of strategies. This follows from the assumption that 10 + 5 is not known as a number fact. If it was known, then ' lookup 7 + 3 followed by lookup 10 + 5' would have been the most efficient combination. The two 2-term rules representing the most efficient combination for this problem are

(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y))
(equal (x + y) 10) ---> *(fn lookup i1 x i2 y)*

(number i3 x) (number i4 y) (not (used i3 x)) (not (used i4 y))
---> *(fn col i3x i4 y).*

5.5 Discussion and further work

5.5.1 Psychological plausibility

Ideally, a model of learning of the mechanisms for transition from one snapshot of human behaviour to the next should be psychologically valid.

There is no claim that the models described in this chapter satisfy this criterion. However, an attempt has been made to make them as plausible as possible. The empirical work was used to guide the algorithms and to match outputs of the models. The prerequisite or domain knowledge, for example the operators for efficiency-driven learning, was compiled from children's protocols. The end-products of the learning models are simulations of children's performance.

The failure-driven learning models a mechanism for the transition from ability to solve 2-term problems to that of 3-term problems. No explanation of this type of transition has been provided in the past. The generalization mechanism described above is one hypothesis for such a transition. In the empirical work, children showed evidence of the use of knowledge of 2-term problems for solving 3-term problems. The transition from 2-term strategies to the 3-term linear strategy was discussed in section 5.3.2 above.

The *dropping adjacency condition* rule for generalization was supported by some children's performance, for example, Craig's protocols from the longitudinal study (chapter 3). The first time that children were observed solving a 3-term problem, most of them used the linear strategy. However, there were also a few children who were observed to use the grouping strategy. Craig's protocols suggest a possibility that the first time children encounter a 3-term problem, they do not necessarily apply the linear strategy. The first two interview sessions during the longitudinal study that Craig was given a 3-term problem, he could not solve it.

E: 4 + 6 + 4 =

Craig: I don't know.

E: I know you know it. Have you ever seen this kind of problem before?

C: No

...

On the next session, less than three weeks later, he used the grouping strategy. It is improbable that Craig encountered a 3-term problem between the interview sessions. It is also improbable that he used the linear strategy during that time, and then changed strategy to grouping.

E: 3 + 4 + 3 =

Craig: 10

E: How did you do it, Craig?

Craig: I knew that 3 and 3 are 6, I left the 4 out until the end.

Furthermore, at the interview sessions when Craig did not know how to solve 3-term problems, he used the 'count all starting from the first addend' strategy (CAF) for solving 2-term problems. On the following session, he used grouping and the 'count all from the larger addend' strategy (CAL). There are two possible explanations for his transition from not being able to solve 3-term problems to solving them using the grouping strategy. Firstly, it could be that he learnt that the order in which the problem was written did not matter, and hence they could be solved in any order. For 3-term problems, 'the problem could be solved in any order' is represented in PALM as 'dropping the adjacency condition' since by dropping this condition, PALM is able to add the three numbers in any order. Secondly, he could have had some sort of 'adjacency' representation of the problem which did not allow him to solve 3-term problems before, and now, 'dropping the adjacency' representation allowed him to solve the problem. With either possible explanation of his cognitive processes, the *dropping adjacency condition* rule models the transition. Further empirical work might be able to support the validity of the transition modelled in PALM. Such empirical work would pay particular attention to those children who are able to solve 2-term problems but are not able to solve 3-term problems, and would be aimed at finding the 'gap' between the two states.

For the efficiency-driven case, the end-products are consistent with the behaviour observed in children. After learning the conditions, PALM's strategies for solving 3-term problems are the grouping strategies that children were observed using. Moreover, the conditions in the learnt rules account for the reasons that children gave for the particular order in which they solved the problem. Even when children gave explanations like "because it is easier to do that first" (and the experimenter did not succeed in persuading the child to make an explicit statement on *why* that particular choice was easier), their choices of the numbers they added first could be explained by the use of operators such as those used for learning in PALM.

In sum, the models of learning in PALM may not necessarily be the only ones that could explain children's cognitive processes in the specific domain of 3-term problem solving. Detailed investigations of other learning mechanisms such as analogy and explanation-based learning, and their applicability to this domain might lead to using them as alternatives for modelling. For example, as a test for the plausibility of learning by analogy, empirical work can be carried out to study whether children use analogy with previous problems. This could be investigated through a sequence of problems; for example,

3 + 4 = 7
3 + 4 + 6 =

3 + 5 = 8
6 + 3 + 5 =

Another specific type of problem where analogy could be applied is on commutative problems like 4 + 7 and 7 + 4. The 'copy answer' strategy for the second of such pairs of problems could be interpreted as the use of analogy. The statements made by children who used this strategy, like "it's the same as that one" suggest the use of analogy.

5.5.2 Failure-driven learning

Rules in PALM are generalized by deleting conditions. Deleting conditions does not necessarily generate rules that produce correct behaviour, but only a rule that is applicable. However, this is not a limitation of the system since generalizations that do not result in correct behaviour may account for children's generalization errors. For example, on 2-term commutative problems, dropping the addition (+) operator accounts for children's generalization of the concept to subtraction. This kind of generalization technique can be used in an ITS for generating possible generalizations that children can make. The correct generalization, from the point of view of the ITS would be the one that matches its student's generalization. The implications of such generalizations for ITS is discussed further in the next chapter.

The empirical investigation in chapter 3 did not resolve how the transition from 2-term to 3-term strategies takes place. Furthermore, it is difficult to investigate this transition empirically since children either know 3-term addition or they do not know it. There isn't anything like 'they know 2 and a 1/2 term...'. The computational model of learning by generalization is one hypothesis for a mechanism for the observed transition. This mechanism and the results of the empirical work can be used as a pilot for designing further empirical work to investigate the transition in more detail and to provide feedback on the model.

There are some similarities between the failure-driven learning in PALM and the impassse-driven learning proposed by VanLehn (1988). Firstly, both methods assume that learning occurs at 'impasses'. Learning occurs when the current knowledge base is insufficient to solve a given problem. VanLehn argues further that it is not just any incompleteness that causes learning. For learning to take place, problem solving must require a piece of knowledge that the problem solver does not possess. In PALM, failure-

driven learning occurs when there are no explicit rules for solving certain types of problems. Secondly, after learning, both in PALM and in the case of impasse-driven learning, the problem solving procedure continues from where it was stuck.

5.5.3 Efficiency-driven learning

Condition learning

As a result of learning the conditions for efficient problem solving, the position of the learned rules in the set of rules is important. The learned, specific rules need to be executed in preference to the more general ones from which they were derived. In PALM, as new rules are learnt, they are added at the the beginning of the list of rules. This is because one of the conflict resolution strategies in PALM is rule ordering. After applying the other resolution strategies, rule ordering selects the first rule, which, as a result of the order of the rules in the ruleset, is one of the most specific rules in the conflict set. If PALM's conflict resolution strategies included rule specificity, then there would not be any restriction on where in the list of rules the new rules that it learns are placed. Rule specificity would select specific rules over more general ones. An immediate improvement on the implementation of PALM would be to include rule specificity as an additional conflict resolution strategy. In PALM, a simple measure of specificity could be the number of conditions in a rule. A rule with more conditions would be more specific than another with fewer conditions. For example, the first rule below is more specific than the second one.

(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y))
(equal (x + y) 10) ---> (fn lookup i1 x i2 y)

(number i1 x) (number i2 y) (not (used i1 x)) (not (used i2 y))
---> (fn col i1 x i2 y).

In the method for finding conditions for grouping two of the numbers in a 3-term problem, only 1 condition has been considered at a time. An obvious extension to this method is to allow the addition of more than one condition. For example, 6 + 4 + 4, satisfies two of the conditions, **(equal 4 4)** and **(equal (6 + 4) 10)**. In this case, learning more than one condition provides a more specific description of this particular type of problems. The choice of which two numbers to add first in such problems can be represented in the action part of the rules.

The idea of finding specific conditions is similar to that of ACM. However, ACM learns by discrimination because it has negative instances as well. In PALM's condition finding process, there are only positive instances, since its focus is on *different* strategies, and not incorrect ones. Furthermore, ACM learns by discriminating positive instances of an operator from its negative instances over a set of examples. PALM, on the other hand considers only one example and one that is a positive instance.

Strategy Composition

Composition is the process of combining multiple steps into one. Strategy composition is the learning process where two rules are combined to make a new one. The composition algorithm in ACT* (Anderson, 1983) can be described as follows. New rules are generated by combining old ones. The conditions of the old rules are combined to form the conditions of the new rule. The actions of the old rules are combined into a sequence of actions in the new rule. The resulting composition is a single new rule that accomplishes the combined effects of the old rules. Anderson (1983) and Neves and Anderson (1981) provide evidence of the psychological plausibility of composition. For example, composition accounts for speedup of a skill with practice. In this section, the application of composition to 3-term strategies as an extension of PALM's efficiency-driven learning is discussed.

In PALM, 3-term problems are solved by applying two 2-term strategies, (or one 2-term strategy twice). PALM could employ an ACT*-like knowledge composition algorithm (Anderson, 1983, 1986) to combine two 2-term strategies that solve 3-term problems. The composition algorithm could be applied in order to make an explicit 3-term strategy. The whole condition side of the combined rule would become a 2-term strategy and the action side would represent the remaining 2-term strategy. This would make PALM's problem solving process more efficient, since it would not have to apply its cycle of rule matching, conflict resolution and executing the selected rule twice. It is also consistent with children's strategies for solving 3-term problems. Although they solve the problem using two 2-term strategies, they do not perform an addition on two terms and then decide to do another 2-term addition. Instead, before they perform any addition, they know how they are going to solve the problem. They know in advance that they are going to perform two 2-term additions.

Furthermore, in the condition-learning mechanism described above, 3-term problems are solved using two separate rules. The strategy used for solving a given problem may not necessarily be the most efficient one, since the overall strategy is not known until the problem is solved. The efficiency of only the first 2-term addition is known. The overall efficiency then depends on the remaining 2-term strategy. However, if the two 2-term strategies were combined as one strategy, then the overall efficiency of the 3-term strategy can be computed. The possible combinations of 2-term strategies would lead to 3-term strategies with different efficiencies. Hence, in choosing the most efficient strategy, the overall efficiencies of the different 3-term strategies can be compared.

In the process of solving a given problem, a pair of rules which are executed one after the other and where the execution of one depends on the results of the other can be composed into one as follows:

R1: C1 ---> A1

R2: C2 ---> A2

R1 & R2: C1 A1 C2 ---> A2

It is A2, the action of the second rule, which is the solution to the given problem. A1 is an intermediate step in the problem solving, the result (or one of the results) of which is used by one (or more) of C2, the conditions of the second rule. In other words, the problem is solved by A2 which depends on the results of A1. The action side of the composed rule contains the actions of the second rule, A2. The condition side consists of all the conditions and actions of the first rule and the conditions of the second rule. The actions of the first rule are needed because combination can only be used when all conditions of both rules are satisfied, and the conditions of the second rules cannot be satisfied without A1. Ideally, instead of containing all the conditions from rule 2, it should contain only the conditions that do not match the actions of the first rule. This is because the conditions in rule 2 that match the actions of rule 1 are redundant. However, they are included in the new rule because the variables in the actions of the second rule (and of the resulting rule) depend on these conditions for their bindings.

The following is an example of two rules that could be composed into one that accomplishes the combined result:

R1: *(number i1 no1) (number i2 no2) (not (used i1 no1)) (not (used i2 no2)) (equal (no1 + no2) 10)* ---> *(fn lookup i1 no1 i2 no2)*

R2: *(number i3 no3) (number i4 partial-ans) (not (used i3 no3)) (not (used i4 partial-ans))* ---> *(fn col i3 no3 i4 partial-ans).*

R1 & R2: *(number i1 no1) (number i2 no2) (not (used i1 no1)) (not (used i2 no2)) (equal (no1 + no2) 10) (fn lookup i1 no1 i2 no2) (number i3 no3) (number i4 partial-ans) (not (used i3 no3))*

(not (used i4 partial-ans)) ---> *(fn col i3 no3 i4 partial-ans).*

The order of the conditions in the new rule is important, since the conditions from rule 2 depend on the actions of the first rule. In the above example, in the combined rule, *(number i4 partial-ans)* is the result of the action *(fn lookup i1 no1 i2 no2).* Hence, to solve 3 + 6 + 4 using the combined rule, *no1* would bind to 6, *no2* to 4, *lookup* would return 10, *partial-ans* would bind to 10, *no3* would bind to 3 and the action would evaluate 10 + 3 using the *col* strategy.

When the action part of the first rule becomes the condition part of the combined rule, it plays the role of a condition as well as an action. The role of an action would be to plan ahead and decide if a certain grouping of addends would leave a more efficient addition for completing the problem solving. For example, for a problem like 5 + 1 + 6, if 6 + 6 is known as a number fact, then, 'planning ahead' could lead to performing 5 + 1 first, since it leaves 6 + 6 as a *lookup.*

Note that Anderson's (1983) mechanism for composition will not work for the task of arithmetic addition. Both the task of arithmetic strategies and of those described by Anderson are sequential. Two rules that represent a sequential task are reduced to one rule that achieves the same task. In Anderson's tasks, the subtask represented by the second rule follows the subtask represented by the first rule, but does not depend on the *results* of the first subtask. In the rules representing the arithmetic problem solving described here, the result of the first step determines the following step. Rule composition in ACT* can be described as follows:

R1: C1 ---> A1

R2: C2 ---> A2

R1 & R2: C1 (C2 - C2A1) ---> A1 A2

where C2A1 represents the conditions of rule 2 that match A1, and (C2 - C2A1) represents the conditions of rule 2 that do not match A1. Note that the conditions of rule 2 that match A1 are not included in the condition side of the new rule, which reveals that the actions of rule 2 do not depend on the actions of rule 1. An example of a task where this kind of composition is applicable is dialing a telephone number. This requires dialing a sequence of numbers. The following two rules illustrate part of the process of dialing the number 65310

> IF the goal is to dial 65310 and 6 is the first number THEN dial 6.
> IF the goal is to dial 65310 and 6 has just been dialed and 5 is after 6 THEN dial 5.

Composition of these two rules would create

> IF the goal is to dial 65310 and 6 is the first number and 5 is after 6 THEN dial 6 and then 5.

Note that the action of dialing 6 does not generate an output or a result that is used for the second action (dialing 5). In contrast, in the task of 3-term addition, the *result* of the first action (sum of two addends) is used as an input for the second action (sum of the 3rd addend and the result of the first action). The proposed mechanism for rule composition in PALM would result in

C1 A1 C2 ---> A2

A1 is performed as a condition of the new rule; A2 depends on the results of A1.

For simplicity, the mechanism above only describes the composition of two rules into one, but the same procedure is easily extendable to composing multiple rules. As learning continues, composed rules can be composed with other rules, until there is a single rule for solving the problem. For example, a rule for solving 4-term problems can be created by composing a 3-term rule with a 2-term rule. ACT* is capable of composing multiple rules - when a goal is completed successfully, all the executed rules are composed together, yielding a single rule that accomplishes the goal (Anderson, 1983).

5.6 Summary

This chapter introduced some of the techniques of learning that have been studied by researchers in machine learning - learning from examples, learning by analogy and explanation-based learning. The choice of the learning methods in PALM was discussed. The implementation of learning models based on existing ones (Michalski's learning by generalization and ACM's condition learning) have been discussed. The chapter described the models applied to children's strategies for solving elementary addition problems. A model of failure-driven learning has been implemented as a candidate mechanism for children's transition from ability to solve 2-term problems to that of 3-term problems. Furthermore, the implementation of an efficiency-driven model of learning that learns to apply children's number facts to problems in order to save work in computing a sum was described. The last section of the chapter discussed strategy composition as an extension of PALM's efficiency-driven learning.

The model of failure-driven learning can be extended to other operators like multiplication and subtraction, and to other domains like fractions and algebra. The efficiency-driven learning can be extended to other domains which involve alternate strategies for problem solving.

The relation of such models of learning to ITS and some directions for further work are discussed in the next chapter.

Chapter 6

CONCLUSIONS

The final chapter presents an overview of the thesis. The contributions of the research are highlighted. The research is discussed in the context of ITS. Finally, some directions for further research are proposed.

6.1 Summary of thesis

The overall intention of this research has been to model the transition from procedural knowledge of commutativity to that of associativity. In order to do this, models of children's strategies for solving problems associated with the two concepts which describe 'snapshots' of their performance at different stages of development were constructed first. Then, learning models were constructed as a candidate mechanism for the transition from one performance level to another. A model of learning by generalization has been constructed for transition from performance on 2-term problems (e.g. 4 + 5) to that on 3-term problems (e.g. 4 + 6 + 3). This model is one hypothesis for the mechanism underlying the observed transition.

Furthermore, a computational model has been constructed for learning the most efficient strategy for solving a given problem. To learn more efficient strategies, PALM learns features of problems to which number facts could be applied. For 3-term problems, specific conditions, that is number facts, are applied to the input problem to select which two of the three numbers to add first. The system begins with general rules representing basic strategies for solving 3-term addition problems, and a set of operators, like (equal x y), representing children's number facts. It learns specific rules by adding operators to the general rules in order to define conditions for efficient problem solving. An ACT*-like (Anderson, 1983) knowledge composition algorithm to combine a sequence of actions into a single task

has also been suggested as further work. Knowledge composition would combine two 2-term strategies to form one 3-term strategy (3-term problems are solved by applying two 2-term strategies).

The production-rule formalism was used for modelling children's strategies on problems related to commutativity and associativity. The models represent 'snapshots' of children's problem solving behaviour. The construction of the models showed features of children's problem solving that were not noticed initially in the studies. For example, while performing an addition, children do not normally count a set of objects more than once. For problems in which two addends are the same, the two addends serve slightly different purposes. Such features needed to be considered in the implementation in order to simulate children's performance. The simulations are in the form of sets of rules, where a set represents problem solving using one of the observed strategies. As in previous production system models, these models do not describe the processes involved in development. However, as described above, the learning component of PALM moves towards modelling the learning process.

The models are based on empirical investigations of children's performance on problems related to the two concepts commutativity and associativity. The thesis reports three studies that were carried out to examine children's strategies for solving problems like 3 + 6 and 3 + 4 + 5, and to study the transition from 2-term to 3-term problems. It discusses the performance levels of commutativity that were identified in the studies. It also discusses children's responses to subtraction problems where the subtrahend is smaller than the minuend.

The pilot study gave indications of the age range in which children should be studied for the concepts and of what tasks could be performed in order to get the most out of the students. The main study provided a space of strategies that children at different levels used. It also proposed

performance levels of the concept of commutativity. Furthermore, the main study investigated the transition from commutativity to associativity. The longitudinal study provided more detailed analysis of children's behaviour, for example, change in strategies over time. It also supported the conclusions obtained in the main study.

To conclude, this research has demonstrated the strength of the combination of the two methodologies of understanding a task: empirical work and computational modelling. Significant steps have been taken towards the goal of modelling learning processes. Although the mechanisms of transition are not necessarily psychologically valid, they are plausible. The algorithms are based on empirical data. The inputs and outputs at different stages in the algorithm are consistent with empirical evidence.

6.2 Contributions

* **Performance levels of commutativity**
* **Estimates of efficiencies of strategies**
* **A candidate mechanism of transition from 2-term to 3-term problem solving**
* **A model of efficiency-driven learning**
* **Generalization errors can be modelled using the model for transition**
* **PALM can be used for modelling in other domains**
* **PALM's learning mechanisms can be used for generating student models**

Performance levels of commutativity

The thesis reported empirical investigations of children's acquisition of commutativity and associativity. Four levels of performance of the concept of commutativity are proposed. Such a detailed investigation of the concept has not been carried out before.

Estimates of efficiencies of strategies

Efficiencies of strategies are estimated based on the amount of work involved and the demand on the child's memory. Although it is a very basic measure, it serves the purpose of distinguishing a more efficient strategy from a less efficient one. The procedure for the efficiency estimates works for 2-term as well as for 3-term problems.

A candidate mechanism of transition from 2-term to 3-term problems

The transition from commutativity to associativity has not been considered at all in previous research. Furthermore, cognitive modelling of the transition from the ability to solve 2-term problems to that of 3-term addition problems has not been undertaken before. The research in this thesis used empirical evidence as well as techniques from machine learning and theories of skill acquisition in order to model a possible mechanism for this transition.

A model of efficiency-driven learning

A model of efficiency-driven learning has been constructed. PALM learns efficient strategies by applying previous knowledge of number facts to given problems. It tests the applicability of the known facts to features of the problem and learns those features that are applicable. PALM applies this mechanism to 3-term addition problems to select two of the three numbers whose sum it already knows as a number fact. Hence it does not have to compute the sum of these two numbers. Furthermore, the model of efficiency-driven learning can be generalized to the acquisition of strategic knowledge in other related domains, for example, algebra.

Generalization errors can be modelled using the model for transition

Children's subtraction errors such as 'smaller from larger' and generalization have not been modelled in the current implementation, but could be included by adding production rules representing such errors.

Generalization of commutativity can also be modelled using the learning by generalization algorithm that has been applied to the transition from 2-term to 3-term problem solving.

PALM can be used for modelling in other domains

The production system used for modelling addition problems can be used for modelling in other similar, closed domains, for example, fractions, algebra and problems with other operators, like multiplication. For example, a rule for finding the common denominator in the fraction domain could be represented in PALM as follows:

(Fraction 1 =N1/=D1) (Fraction 2 =N2/=D2) ---> (fn MULTIPLY =D1 =D2)

The algebra malrule N + X = P => X = P + N could be represented as follows:

(lhs Number =N Variable =X) (rhs Number =P) ---> (lhs Variable =X)
(rhs Number =P Number =N)

To include operations other than addition, the description language could be revised to include the *operator* as well. Hence, a multiplication problem could be described as follows:

(Number 1 =X) (Number 2 =Y) (Operator *) ---> (fn * =X =Y)

PALM's learning mechanisms can be used for generating student models

Previous research has used machine learning techniques for automating the construction of student models. Examples of such systems are ACM (Langley, Ohlsson and Sage, 1984) and PIXIE (Sleeman, 1983). The main aim of this type of application has been to avoid extensive libraries of 'bugs' by providing some data and getting the machine to generate students' 'bugs'. The resulting student models are static models like those of LMS (Sleeman and Smith, 1981) and DEBUGGY (Brown and Burton, 1978). The efficiency-driven learning component of PALM learns conditions for the

applicability of its previous knowledge, which achieves a similar end result to that of ACM. Hence, the efficiency-driven learning model can be used for constructing student models. PALM has demonstrated this possibility for 2-term and 3-term addition problems. Note that ACM and PIXIE model 'bugs' whereas PALM models a variety of strategies. With domains in which 'bugs' are commonly observed, PALM's efficiency-driven learning mechanism would be able to model 'bugs' once a set of 'buggy' operators such as the 'smaller from larger' operator in ACM is available.

6.3 Implications for ITS

6.3.1 Production-rule models

The production-rule models in PALM can be used for student modelling in an ITS, like those previous tutoring systems that have been implemented in production systems: for example, for teaching medicine (Clancey, 1982), programming skills (Anderson and Reiser, 1985), for geometry (Anderson, Boyle and Yost, 1985), and for quadratic equations (O'Shea, 1979). In order to diagnose the students' strategies, an interface, like the Graphical Arithmetic Description Language (Evertsz, Hennessy and Devi, 1988) will be needed for the students to simulate their arithmetic problem solving and to communicate it to the ITS. The Graphical Arithmetic Description Language (GADL) interface was designed for children to explain their informal arithmetic strategies to a student modelling system and similarly for the system to communicate example solutions to its users. The information gathered from such an interface could be compared to the production-rule models, which could then be used as a basis for tutoring. For example, the tutor could use the knowledge of the estimated efficiencies of strategies to guide its students to use more efficient ones.

Similarly, an interface like GADL could be used for subtraction problems. The interface could be used for distinguishing between generalization of

commutativity and using the 'smaller from larger bug'. Note that the final solution to a subtraction problem using these two strategies is the same. With such an interface, children can display their methods for arriving at their solutions to subtraction problems. Hence, it would also be appropriate for modelling the different reasons for children offering '0' or 'don't know' as their response to problems like 2 - 5.

6.3.2 Conceptual knowledge

As discussed in chapter 4, production-rule models represent knowledge of procedures but do not take adequate account of conceptual knowledge. For example, in solving 5 + 8, a child's strategy of starting from 8 and counting on 5 is a display of his/her procedural knowledge. The related conceptual knowledge would be the concept of commutativity, i.e. 'it does not matter whether one starts counting from the first addend or from the second, the sum is the same'. An ITS that has a production-rule representation for procedural knowledge must have separate machinery to represent the conceptual knowledge. This machinery could involve modelling of tasks that were presented to the students in the empirical work, and the measures that were used to categorise the different levels of the concept of commutativity (Devi, 1990a). For example, a child who generalizes commutativity to subtraction, indicates that s/he possesses some knowledge of the concept. Furthermore, whether a child copies the previous answer on a task like

$$47 + 58 = 104$$

$$58 + 47 = ?$$

provides an indication of whether s/he possesses the knowledge of commutativity or not. Further details of such tasks are presented in chapter 3.

There is evidence from literature concerning domains in which procedural knowledge is acquired before conceptual knowledge (Baroody and Gannon, 1984; Briars and Sieglar, 1984; Fuson, Secada and Hall, 1983), and domains where the reverse is true (Gelman and Gallistel, 1978, Gelman and Meck, 1983, 1987; Greeno, Riley and Gelman, 1984). From our experience of the empirical work, in the domain of arithmetic, it is difficult to establish which comes first. Perhaps further research in the field, including studies aimed at interrogating students to elicit their conceptual knowledge, will help us to improve our understanding of this dilemma. From what is known so far, we conclude that the important thing is that children should possess both, and know the relationship between the two. Hence, the separate diagnosis of conceptual and procedual knowledge could be used by a tutor to help students to link the two types of knowledge, and to facilitate their understanding of their procedures.

6.3.3 Models of learning

Previous student models have not embodied explanations of how a student arrives at a particular knowledge state. This limitation has been noted by other researchers (Brown and Burton, 1978; Hennessy, 1990; Laurillard, 1990; Wenger, 1987). A tutoring system that has the potential of explaining how a student reaches a certain knowledge state, can focus its tutoring on the student's underlying learning. The modelling of learning is a step towards predicting how a student arrives at an answer. For example, in the case of efficiency-driven learning, an ITS would be able to predict not only what two numbers a child added first, but also why s/he chose them. For the problem, 4 + 7 + 4, the learning model would employ the number fact *(equal x y)* for transition from linear to grouping strategy. Using this information, if a student did not use the grouping strategy on such a problem, to facilitate transition to this strategy, the tutor could focus its tutoring on the number fact.

With access to models of transition such as that for the transition from 2-term problems to 3-term problems, the tutor, in addition to predicting the student's solution, can also predict how that student got to that stage. With such additional information about the student's learning strategy, the tutor can predict the learning outcome before a task is presented to the student. This would provide the tutor with further information on which to base its teaching actions. Furthermore, once the tutor has information about the student's learning outcome in addition to how that outcome was achieved, it has more information for its further tutoring.

The tutoring knowledge could be represented in the system as static knowledge. This could take the form of production rules with the left-hand side representing a learning strategy and the right-hand side representing a corresponding tutoring strategy. Examples of such tutorial rules are:

> IF learning by dropping condition generalization rule achieves the desired state, THEN present a sequence of examples to facilitate this generalization.
>
> IF learning by generalization, THEN present a counter-example.

If more than one such tutorial rule could be applied at a particular time, then conflict resolution strategies, which could be based on information from the student's past history, could be employed.

The bounded user modelling technique (Elsom-Cook, 1987) is an example of dynamic modelling. Since it is very difficult to construct a model of the student's exact knowledge state, Elsom-Cook proposed that the upper and lower bounds of the student's possible knowledge state could be constructed, such that the exact model lies somewhere in between. He proposed that the technique can lead towards making predictions of the student's learning process. The bounded user modelling technique can be used in an ITS for representing children's specialization and generalization of the concept of

commutativity. An example of specialization of commutativity is its application to small numbers only: there are children for whom 10 + 7 is obviously the same as 7 + 10, but they do not recognize the concept for 70 + 100 and 100 + 70. Given operators like (< 10 (x + y)) to model specialization, the learning model can add such conditions to the rules representing the student's application of the concept. Such conditions can eventually be dropped to model generalization of the concept to all numbers. Similarly, children's generalization of the concept to subtraction can be modelled. Furthermore, the specialization algorithm for learning to apply number facts to problem solving, as well as the generalization algorithm for modelling transition from 2-term problem solving to 3-term problem solving (detailed in chapter 5) can be applied in the bounded user modelling technique to define the 'space' of models within which a student's strategy lies.

Finally, there is little research concerning teaching strategies in ITS. Two systems that have used teaching strategies are PROTO-TEG (Dillenbourg, 1988) and DOMINIE (Elsom-Cook and Spensley, 1987; Elsom-Cook, 1989). They represent only the student's current knowledge state. They do not take into account how that state was arrived at. Further work on teaching strategies is dependent on dynamic student modelling. More specifically, it is dependent on learner-based models like PALM. PALM demonstrates the simplest type of student model that can be used with respect to teaching strategies. Dynamic modelling would enable a tutoring system to make more informed predictions about teaching strategies it should use. For example, the system could have knowledge about teaching strategies associated with learning strategies. Once the system has a mechanism for transition from the current state to a desired state, it can choose the teaching strategy accordingly.

6.4 Further work

The immediate next steps in relation to the present research should be to improve on the psychological validity of the learning mechanisms by carrying out further and more detailed empirical studies related to the models. This section outlines some proposals for further research concerning empirical work and computational modelling.

6.4.1 Empirical Work

In the empirical studies reported in this thesis, commutativity and associativity of addition only were investigated. Related empirical research could include the extensions of the concepts to the other operators of arithmetic, and to other domains like fractions and algebra.

Furthermore, it is generally believed that children's invention of informal algorithms is based on principles like commutativity. The empirical studies in the current research showed evidence of children using algorithms and not possessing the underlying conceptual knowledge. It will be interesting to investigate if informal algorithms are invented as a result of their underlying conceptual knowledge or not.

A major issue that remains to be answered regarding the concept of commutativity is how children come to learn it and how they learn to apply it. Baroody and Gannon (1984) propose that it is learnt by discovery. Other possible means of learning this concept are from abstract examples, and from experience with real situations, for example, calculations using sweets (like Smarties). There is scope for investigating whether the concept can be taught or not, and if so, then how.

The models of learning in PALM produce human-like behaviour that is compatible with the empirical studies. In order to improve the validity of the mechanisms of learning, empirical tasks could be defined to study

specific parts of the mechanisms in detail. This could include evaluation of specific parts of the learning mechanisms. For example, detailed studies could be carried out to study the transition from commutativity to associativity. The empirical tasks on which the models are based were general - they included 2-term problems, 3-term problems, subtraction, levels of development of commutativity, and so on. As a result of the implementation, more specific tasks can be designed for further investigation. The description language, for example, the concept of 'adjacency' can be tested for its validity. Tasks can be designed to investigate whether children apply analogy, generalization, etc. in specific cases. Such evaluation, and comparison of human learning and the program's performance can lead to improvements in the cognitive models.

6.4.2 Computational modelling

As outlined above, the psychological validity of the learning mechanisms can be improved through empirical testing. Psychologically valid and more detailed models of learning need to be developed. One of the limitations of the models is that they do not take conceptual knowledge into account. An obvious suggestion for further work involves incorporating this type of knowledge.

The model of transition from 2-term problems to 3-term problems is only one possible hypothesis for the transition. Other types of learning, for example, learning by analogy, explanation-based learning, and learning from examples should also be considered.

The techniques of learning by generalization and by discrimination have been employed for cognitive modelling by other researchers; for example, in ACT* (Anderson, 1983) and in SOAR (Rosenbloom and Newell, 1986). Further evaluation of the models in PALM can be carried out by comparing the performance of the program with that of such systems.

6.5 Summary

This chapter presented a summary of the thesis. It highlighted the contributions of the research reported in the thesis. The importance of the work in relation to ITS was discussed. The thesis concluded by outlining some areas for further work.

In sum, the research reported in this thesis demonstrates the feasibility of implementing computer models of certain aspects of mechanisms of transition from one stage to another. The iteration of empirical work and computational modelling provides a promising approach towards psychological models of learning.

REFERENCES

Ahn, W., Mooney, R., Brewer, W. F. & DeJong, G. F. (1987). Schema acquisition from one example: Psychological evidence for explanation-based learning. *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*, 50 - 57.

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1986). Knowledge compilation: the general learning mechanism. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 2. Los Alto, CA: Morgan Kaufmann.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*, 94 (2), 192 - 210.

Anderson, J. R. (1989). A theory of the origins of human knowledge. *Artificial Intelligence*, 40, 313 - 351.

Anderson, J. R., Boyle, C. F. & Yost, G. (1985). The geometry tutor. *Proceedings of the International Joint Conference on Artificial Intelligence*, Los Angeles, 1 - 7.

Anderson, J. R. & Reiser, B. J. (1985). The LISP tutor. *Byte*, 10 (4), 159 - 175.

Ashlock, R. B. (1982). *Error patterns in computation: a semi-programmed approach*, third edition. Columbus, Ohio 43216: Charles E. Merrill Publishing Co.

Baroody, A. J. (1984). The case of Felicia: A young child's strategies for reducing memory demands during mental addition. *Cognition and Instruction*, 1 (1), 109 - 116.

Baroody, A. J. (1985). Mastery of basic number combinations: internalization of relationships or facts? *Journal for Research in Mathematics Education*, 16 (2), 83 - 98.

Baroody, A. J. & Gannon, K. E. (1984). The development of the commutativity principle and economical addition strategies. *Cognition and Instruction*, 1 (3), 321 - 339.

Baroody, A. J. & Ginsburg, H. P. (1982). Generating number combinations: rote process or problem solving? *Problem Solving*, 4(12), 3 - 4.

Baroody, A. J. & Ginsburg, H. P. (1986). The relationship between initial meaningful and mechanical knowledge of arithmetic. In Hiebert, J. (Ed.), *Conceptual and procedural knowledge: the case of mathematics.* London: Erlbaum.

Baroody, A. J., Ginsburg, H. P. & Waxman, B. (1983). Children's use of mathematical structure. *Journal for Research in Mathematics Education*, 14 (3), 156 - 168.

Briars, D. & Siegler, R. (1984). A featural analysis of preschoolers' counting knowledge. *Developmental Psychology*, 20, 607 - 618.

Brown J. S. & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2, 153 - 192.

Brown, J. S. & VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4, 379 - 426.

Brown, J. S. & VanLehn, K. (1982). Toward a generative theory of bugs. In Carpenter, Moser and Romberg (Eds.), *Addition and subtraction: a cognitive perspective.* Hillsdale, NJ: Erlbaum.

Brownston, L. Farrell, R. Kant, E. & Martin, N. (1985). *Programming expert systems in OPS5: an introduction to rule-based programming.* Massachusetts: Addison-Wesley.

Carbonell, J. G. (1983). Learning by analogy: formulating and generalizing plans from past experience. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 1. Los Alto, CA: Morgan Kaufmann.

Carbonell, J. G. (1986). Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J.B. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol. 2. Los Alto, CA: Morgan Kaufmann.

Carpenter, T. P. (1986). Conceptual knowledge as a foundation for procedural knowledge. In Hiebert, J. (Ed.), *Conceptual and procedural knowledge: the case of mathematics*. London: Erlbaum.

Carpenter, T. P. & Moser, J. M. (1983). The acquisition of addition and subtraction concepts. In R. Lesh & M. Landau (Eds.), *Acquisition of mathematics concepts and processes*. New York: Academic Press.

Clancey, W. J. (1982). Tutoring rules for guiding a case method dialogue. In Sleeman, D. H. & Brown, J. S. (Eds.), *Intelligent Tutoring Systems*. London: Academic Press.

Clancey, W. J. (1986). Qualitative student models. In Traub, J. F. (Ed.), *Annual Reviews of Computer Science*, 1, 381 - 450.

DeJong, G. & Mooney, R. (1986). Explanation-based learning: an alternative view. *Machine Learning*, 1 (2), 145 - 176.

Denvir, B. & Brown, M. (1986). Understanding of number concepts in low attaining 7 - 9 year olds: part 1, Development of descriptive framework and diagnostic instrument. *Educational Studies in Mathematics*, 17, 15 - 36.

Devi, R. (1989). *Machine Learning and Tutoring Systems*, Technical Report No. 61, Center for Information Technology In Education, The Open University.

Devi, R. (1990a). Acquisition of commutativity and associativity. *Proceedings of the weekend conference, British Society for Research into Learning Mathematics (BSRLM)*, Thames Polytechnic, London, 37 - 43.

Devi, R. (1990b). Modelling acquisition of commutativity and associativity. *The 7th International Conference on Technology and Education (ICTE)*, Brussels, Belgium, 438 - 440.

Devi, R. (in press). Modelling children's arithmetic strategies. In Elsom-Cook & Moyse, R. (Eds.), *Knowledge Negotiation*. London: Paul Chapman.

Devi, R., O'Shea, T., Hennessy, S. & Singer, R. (in press). Modelling informal arithmetic strategies. To appear in Laborde, J. M. (Ed.), *Student modelling: the case of Geometry*, Grenoble, France.

Dietterich, T. G. & Michalski, R. S. (1983). A comparative review of selected methods for learning from examples. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 1. Los Alto, CA: Morgan Kaufmann.

Dillenbourg, P. (1988). A pragmatic approach to student modelling: Principles and architecture of Proto-Teg. *Proceedings of Intelligent Tutoring Systems*, LeMans.

Donaldson, M. (1978). *Children's minds*. Glasgow: Fontana.

Elio, R. & Anderson, J. R. (1981). Effects of category generalizations and instance similarity on schema abstraction. *Journal of Experimental Psychology: Human Learning and Memory*, 7, 397 - 417.

Elsom-Cook, M. (1984). *Design considerations of an intelligent tutoring system for Lisp*. Unpublished PhD thesis, Department of Psychology, University of Warwick.

Elsom-Cook, M. (1987). *Guided discovery tutoring and bounded user modelling in Intelligent Computer Aided Instruction.* Technical report no. 13, Center for Information Technology In Education, Institute of Educational Technology, Open University.

Elsom-Cook, M. & Spensley, F. (1987). Using multiple teaching strategies in an ITS. In *Proceedings of ITS-88*, Montreal.

Elsom-Cook, M. (1989). *Dialogue and teaching styles.* Technical report no. 62, Center for Information Technology In Education, Institute of Educational Technology, Open University.

Evertsz, R. (1991). *The role of the crucial experiment in student modelling.* Unpublished PhD thesis, Institute of Educational Technology, Open University.

Evertsz, R. & Elsom-Cook, M. (1990). Generating critical problems in student modelling. In Elsom-Cook, M. (Ed.), *Guided Discovery tutoring: A framework for ICAI research.* London: Paul Chapman.

Evertsz, R., Hennessy, S. & Devi, R. (1988). *GADL: A graphical interface for mental arithmetic algorithms.* Technical Report No. 49, Center for Information Technology In Education, The Open University.

Flann, N. & Dietterich, T. (1986). Selecting appropriate representations for learning from examples. *Proceedings of the National Conference on Artificial Intelligence.* American Association for Artificial Intelligence.

Fuson, K. C. (1982). An analysis of the counting-on solution procedure in addition. In T. P. Carpenter et al. (Eds.), *Addition and Subtraction: A cognitive perspective.* Hillsdale, NJ: Erlbaum.

Fuson, K. C., Secada, W. G. & Hall, J. W. (1983). The transition from counting all to count on in addition. *Journal for Research in Mathematics Education*, 14, 47 - 57.

Gelman, R. (1972). Logical capacity of very young children: Number invariance rules. *Child Development*, 43, 75 - 90.

Gelman, R. (1977). How young children reason about small numbers. In N. J. Castellan et al. (Eds.), *Cognitive theory*, Vol. 2. Hillsdale, NJ: Erlbaum.

Gelman, R. (1982). Basic numerical abilities. In R. J. Sternberg (Ed.), *Advances in the psychology of intelligence*, Vol. 1, 181 - 205. Hillsdale, NJ: Erlbaum.

Gelman, R. & Gallistel, C. R. (1978). *The child's understanding of number.* Cambridge, MA.: Harvard University Press.

Gelman, R. & Meck, E. (1983). Preschoolers' counting: principle before skill. *Cognition*, 13, 343 - 359.

Gelman, R. & Meck, E. (1986). The notion of principle: the case of counting. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: the case of mathematics*. Hillsdale, NJ: Erlbaum.

Gelman, R., Meck, E. & Merkin, S. (1986). Young children's numerical competence. *Cognitive Development*, 1 (1), 1 - 29.

Goldstein, I. P. (1982). The genetic graph: a representation for the evolution of procedural knowledge. In Sleeman, D. H. & Brown, J. S. (Eds.), *Intelligent Tutoring Systems*. London: Academic Press.

Greeno, J. G., Riley, M. S., & Gelman, R. (1984). Conceptual competence and children's counting. *Cognitive Psychology*, 16, 94 - 153.

Groen, G. J. & Parkman, J. M. (1972). A chronometric analysis of simple addition. *Psychological Review*, 79 (4), 329 - 343.

Hartley, J. (1973). The design and evaluation of an adaptive teaching system. *International Journal of Man-Machine Studies*, 5 (2), 421 - 436.

Hayes-Roth, F. & McDermott, J. (1977). Knowledge acquisition from structural descriptions. *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)*, Cambridge, MA., 356 - 362.

Hennessy, S. (1986). *The role of conceptual knowledge in the acquisition of arithmetic algorithms*. Unpublished PhD thesis, University of London.

Hennessy, S. (1990). Why bugs are not enough. In Elsom-Cook, M. (Ed.), *Guided discovery tutoring: A framework for ICAI research*. London: Paul Chapman.

Hiebert, J. & Lefevre, P. (1986). Conceptual and procedural knowledge in mathematics: an introductory analysis. In Hiebert, J. (Ed.), *Conceptual and procedural knowledge: the case of mathematics*. London: Erlbaum.

Hughes, M. (1981). Can preschool children add and subtract? *Educational Psychology*, 1 (3), 207 - 219.

Hunt, E. B., Marin, J. & Stone, P. J. (1966). *Experiments in induction*. New York: Academic Press.

Ilg, F. & Ames, L. B. (1951). Developmental trends in arithmetic. *Journal of Genetic Psychology*, 79, 3 - 28.

Keane, M. (1988). *Analogical methods in concept learning from examples*. Technical report no. 39, Human Cognition Research Laboratory, The Open University.

Klahr, D., P. Langley & R. Neches (Eds.), (1987). *Production system models of learning and development*. Cambridge, MA: The MIT Press.

Klahr, D. & Wallace, J. G. (1976). *Cognitive development: an information processing view*. New Jersey: Erlbaum.

Langley, P. (1983). Exploring the space of cognitive architectures. *Behaviour Research Methods and Instrumentation*, 15 (2), 289 - 299.

Langley, P. (1987). A general theory of discrimination learning. In D. Klahr, P. Langley and R. Neches (Eds.), *Production system models of learning and development*. Cambridge, MA: The MIT Press.

Langley, P. & Carbonell, J. G. (1984). Approaches to machine learning. *Journal of the American Society for Information Science*, 35 (5), 306 - 316.

Langley, P., Ohlsson, S. & Sage, S. (1984). *Machine learning approach to student modelling*. Technical report CMU-RI-TR-84-7. Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Laurillard, D. (1990). The pedagogical limitations of generative student models. In M. Elsom-Cook (Ed.), *Guided discovery tutoring: a framework for ICAI research*. London: Paul Chapman.

Lebowitz, M. (1986a). Concept learning in a rich input domain: generalization based memory. In R. S. Michalski, J.B. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol. 2. Los Alto, CA: Morgan Kaufmann.

Lebowitz, M. (1986b). UNIMEM, a general learning system: an overview. *Proceedings of the 7th European Conference on Artificial Intelligence*, 1, 32 - 42.

Luchins, A. S. (1942). Mechanizing problem solving. *Psychological Monographs*, 54 (248).

Matz, M. (1982). Towards a process model for high school algebra. In Sleeman, D. H. & Brown, J. S. (Eds.), *Intelligent Tutoring Systems*. London: Academic Press.

Michalski, R. S (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 1. Los Alto, CA: Morgan Kaufmann.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule-learning. *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)*, Cambridge, MA., 305 - 310.

Mitchell, T. M. (1978). *Version spaces: an approach to concept learning.* Unpublished PhD thesis, Stanford University.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18, 203 - 226.

Mitchell, T. M., Keller, R. M. & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: a unifying view. *Machine Learning*, 1 (1), 47 - 80.

Neches, R. (1987). Learning through incremental refinement of procedures. In D. Klahr, P. Langley and R. Neches (Eds.), *Production system models of learning and development.* Cambridge, MA: The MIT Press.

Neches, R., Langley, P. & Klahr, D. (1987). Learning, development and production systems. In D. Klahr, P. Langley and R. Neches (Eds.), *Production system models of learning and development.* Cambridge, MA: The MIT Press.

Neves, D. M. & Anderson, J. R. (1981). Knowledge compilation: mechanisms for the automatization of cognitive skills. In Anderson, J. R. (Ed.), *Cognitive skills and their acquisition.* Hillsdale, New Jersey: Erlbaum.

Newell, A. & Simon, H. A. (1972). *Human problem solving.* NJ: Prentice-Hall.

Ohlsson, S. & Rees, E. (1988). *An information processing analysis of the function of conceptual understanding in the learning of arithmetic procedures*. Tech. report no. KUL-88-03, Pittsburgh.

O'Shea, T. (1979). *Self-Improving Teaching Systems*, Basel, Birkhauser.

Payne, S. J. & Squibb, H. R. (1988). *Understanding algebra errors: the psychological status of mal-rules*. CERCLE technical report no. 43, University of Lancaster.

Plotzner, R, Spada, H., Stumpf, M. & Opwis, K. (1990). *Learning qualitative and quantitative reasoning in a microworld for elastic impacts*. Research report no. 59. Psychological Institute, University of Freiburg, West Germany.

Quinlan, R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 1. Los Alto, CA: Morgan Kaufmann.

Resnick, L. B. (1980). The role of invention in the development of mathematical competence. In R. H. Kluwe & H. Spada (Eds.), *Developmental models of thinking*. New York: Academic Press.

Resnick, L. B. (1983). A developmental theory of number understanding. In H. P. Ginsburg (Ed.), *The development of mathematical thinking*. London: Academic Press.

Resnick, L. B. & Ford, W. W. (1984). *The psychology of mathematics for instruction*. London: Erlbaum.

Resnick, L. B. & Groen, G. J. (1977). Can preschool children invent addition algorithms? *Journal of Educational Psychology*, 69, 645 - 652.

Rosenbloom, P. S. & Newell, A. (1986). The chunking of goal hierarchies: A generalized model of practice. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 2. Los Alto, CA: Morgan Kaufmann.

Self, J. (1974). Student models in computer-aided instruction. *International Journal of Man-Machine Studies*, 6, 261 - 276.

Self, J. (1988). Bypassing the intractable problem of student modelling. *Proceedings of ITS-88*, Montreal.

Self, J. & Gilmore, D. J. (1988). The application of machine learning to intelligent tutoring systems. In Self, J. (Ed.), *Artificial Intelligence and Human Learning*, London: Chapman and Hall.

Silver, E. A. (1986). Using conceptual and procedural knowledge: A focus on relationships. In Hiebert, J. (Ed.), *Conceptual and procedural knowledge: the case of mathematics*. London: Erlbaum.

Simon, H. A. (1985). Information-processing theory of human problem solving. In Aitkenhead, A. M. & Slack, J. M. (Eds.), *Issues in cognitive modelling*. Hillsdale, New Jersey: Erlbaum.

Sleeman, D. H. (1983). Inferring student models for intelligent computer-aided instruction. In R. S. Michalski, J. B. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol 1. Los Alto, CA: Morgan Kaufmann.

Sleeman, D. H. (1984). An attempt to understand students' understanding of basic algebra. *Cognitive Science*, 8, 387 - 412.

Sleeman, D. H. & Smith, M. J. (1981). Modelling students' problem solving. *Artificial Intelligence*, 16, 171 - 187.

Spada, H., Stumpf, M., & Opwis, K. (1989). The constructive process of knowledge acquisition: student modelling. In H. Maurer (Ed.), *Proceedings of the 2nd International Conference on Computer-Assisted Learning*, 486 - 499. Berlin: Springer.

Starkey, P. & Gelman, R. (1982). The development of addition and subtraction abilities prior to formal schooling in arithmetic. In T. P. Carpenter et al. (Eds.), *Addition and subtraction: A cognitive perspective*. Hillsdale, NJ: Erlbaum.

VanLehn, K. (1983). *Felicity conditions for human skill acquisition: Validating an AI-based theory*. (Tech. report CIS-21). Xerox Palo Alto Research Center.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, 31 (1), 1 - 40.

VanLehn. K. (1988). Towards a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems*. N. Y.: Springer.

Vere, S. A. (1977). Induction of relational productions in the presence of background information. *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)*, Cambridge, MA., 349 - 355.

Wenger, E. (1987). *Artificial intelligence and tutoring systems, computational and cognitive approaches to the communication of knowledge*. Inc., CA: Morgan Kaufmann.

Winston, P. H. (1975). Learning structural descriptions from examples. In Winston, P. H. (Ed.), *The psychology of computer vision*. New York: McGraw Hill.

Young, R. M. (1976). *Seriation by children: An artificial intelligence analysis of a piagetian task*. Basel, Birkhauser.

Young, R. & O'Shea, T. (1981). Errors in children's subtraction, *Cognitive Science*, 5, 153 - 177.

APPENDICES

Appendix 1. List of problem pairs for task 1

1+2	1+4
1+3	3+1
3+4	4+3
2+5	5+2
6+1	6+1
5+2	2+3
5+4	4+5
3+1	0+1
4+3	1+3
2+3	1+1
3+2	2+3
7+2	3+2
2+1	2+8
2+2	2+2
3+6	6+3
6+2	2+6
2+5	2+10
2+7	10+7
6+4	4+6
3+6	6+4
3+7	7+3
8+2	2+8

Appendix 2. A sample protocol in task 2

(Ex refers to the experimenter)

...

Ex: (writes the problem and reads it out) 4 + 6 =

SE: (slides out 4 blocks from the pile, counting as he does this) 1,2,3,4. (slides them all together on to the left hand side, in front of the pile, does the same for 6, slides them together to the right hand side of the set of 4 (note the sets are not in linear arrays and also the 2 sets are kept separate), then counts them all from the left) 1, 2, 3, 4, ...,10.

(7 + 2 = 9)

Ex: 9 + 6 = ...

SE: (slides the resulting set (9) from the previous problem), this is 9 already, I don't have to pick them up. (takes 6 from the pile and counts the whole set) 1, 2, 3, ..., 15.

(5 + 3 = 8; 2 + 7 = 9; 6 + 4 = 10)

Ex: 6 + 9 ...

SE: 6 + 9. 1, 2, 3, 4, 5, 6, 7, 8, 9. (pause) Oops! (looks at the problem) 6. (counts out 6 from the counted 9; counts 1, 2, 3 for the remaining 3, and 6 from the pile, puts the sets side by side) 1, 2, ..., 15.

Ex: "I'll give you some hard ones now. 7 + 13

SE: Oh! 1, 2, 3, 4, is 5 that how you do your 7? I thought it was something different.

Ex: How do you do your 7?

SE: I do them like that (pointing to one of my previous 7s)

Ex: okay

SE: 1, 2, 3, 4, 5, 6, 7, I'm gonna use these, pick this up (picks up one of the rods, i.e. one with 10 units, starts sliding units from the pile, next to the rod) 11, 12, 13, 14, 15.

(ex asks if SE can do the problems without the counters; he says he cannot; back to the problem)

SE: (counts out 1, 2, ..., 22).

Ex: You sure its the right problem? 7 + 13? Check that you've got the right problem.

SE: 1, 2, 3, ..., 13 (realized that he had 15 instead of 7; Note that he checks 13 before 7, paying no attention to the order). 1, 2, 3, ..., 20.

Ex: good, can you do 13 plus 7?

SE: Have I just done 13 + 7?

Ex: You've just done 7 + 13

SE: ok (gets the 13 set to the front, and the 7 set to the right, counts them all out again)

Ex: can you do 7 plus 14? (writing out 7 + 14)

SE: (counts out 7, counts out 14, then counts them all) 1, 2, ..., 21.

Ex: 21, good. Can you do 14 + 7? You just did 7 + 14.

SE: (Swaps the two sets of counters around and then counts out the whole set) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 (pause) 15, 16, 17, 18, 19, 20 (misses out 1 in between)

Ex: what is it?

SE: 20!

Ex: I don't think that's right.

SE: 1, 2, 3, ... , 21.

Ex: So what is the right answer, 20 or 21?

SE: 20

Ex: why?

SE: because it is (not aware of the similarity of this problem to the previous one)

Ex: 14 + 7?

SE: 21

Ex: ok Can you do 6 + 9?

SE: 1, ..., 6. 1, ..., 9. 1, ..., 15.

(7 + 9 = 16; 9 + 7 swaps the counters around and counts all again)

Ex: 4 + 6

SE: 1, 2, 3, 4. 1, 2, 3, 4, 5, 6. 1, 2, ..., 10.

Ex: Now I'll give you the problem 6 + 4. Can you tell me whether this will be the same or different? will be 10 or not? You just did 4 add 6. Can you tell me if 6 add 4 whether it will be the same (pointing to the 10 above, answer to previous problem) or different?

SE: different

Ex: why will it be different?

SE: Umm... (pause ~ 30 sec) Because they are the other way around.

Ex: So it will be different?

SE: yeah!

Ex: Can you work out 6 + 4 and see if it will be different or not?

SE: ok 1, 2, 3, 4, 5, 6. 1, 2, 3, 4. 1, 2, ..., 10! (a little surprised, looks at the ex for response). So it is the same.

Ex: yes

SE: Ooh!

Ex: So what do you think? It's the same or different?

SE: different (laughs)

Ex: why do you think it's different? Can you tell me why you think 4 + 6 is different from 6 + 4?

SE: because they are the other way around.

Ex: ok, last problem: Do you think the sum of 8 and 2 (writes down 8 + 2) and the sum of 2 and 8 (writes down 2 + 8) are different?

SE: different

Ex: why are they different?

SE: because they are the other way around.

Appendix 3. A sample of problems in the main study

4 + 5

5 + 4

7 + 9

9 + 7

8 + 7

7 + 8

6 + 10

10 + 6

7 + 13

13 + 7

11 + 14

14 + 11

4 - 2

2 - 4

Word problem: Suppose you bought lollies worth 11c, 7c and 8c. How would you find out how much money you needed to give to the shopkeeper? Can you write down the problem?

2 + 5 + 5

2 + 6 + 4

6 + 3 + 7

3 + 8 + 8

10 + 16 + 10

Appendix 4. Table showing details of subjects and results of the main study on commutativity stages, generalization to subtraction and 3-addend addition

ability is one of high, medium or low (determined by the teacher).

subtraction column: G for generalization, SFL - smaller from larger, 0 - zero, NP - not possible, - for those not tested.

grouping: 1 - grouping, 2 - explicit knowledge of grouping but did not use it, 3 - applied comm. to first 2 terms, 4 - no evidence of transfer, 5 - did not know comm. but used COL, 6 - left to right strategy (did not know comm.).

other operator: marks the subjects tested for their performance on other operators.

subject	age	ability	comm. stage	subtraction	grouping	other operator
1. AAK	6.6	H	ii	G	6	
2. DR	6.4	H	iii	NP	4	
3. STM	6.3	H	ii	G	6	
4. JMS	6.8	H	i	NP	6	
5. AAN	6.9	H	ii	NP	6	
6. KD	6.8	H	ii	-	-	
7. MC	6.11	H	i/ii	0	-	
8. YPR	6.3	H	iv	-	1	
9. AS	6.10	H	iii	NP	4	
10. AS	6.10	H	iv	NP	1	
11. AK	6.8	H	iv	0	1	
12.AN	6.7	H	iii	G	2	
13.MSN	6.11	M	iii	0	2	
14.VS	6.6	H	iv	G	1	
15.RDP	6.5	H	iv	0	1	
16.AAN	6.10	M	iv	NP	1	
17.RDD	6.5	M	i	-	6	
18.AS	6.3	M	iv	0	1	

19.PR	6.9	M	i	-	6
20.MS	6.6	L	i	-	-
21.PS	6.7	L	i	-	-
22.PA	6.4	L	i	-	-
23.AD	6.9	M	i	-	-
24.JS	6.11	H	iv	-	1
25.SP	6.8	M	iv	0	1
26.RM	6.6	M	iv	G	1
27.AK	6.8	M	ii/iii	G	-
28.AS	5.8	H	i	-	-
29.M	5.7	M	i	-	-
30.S	6.6	L	ii	-	-
31.AK	5.9	L	i	-	-
32.K	6.8	L	ii	-	-
33.WA	6.9	H	ii	0	-
34.PA	6.3	H	ii	0	-
35.SB	5.9	M	i	-	-
36.APS	7.6	H	iv	G	1
37.AS	7.6	H	iv	SFL	1
38.NV	7.8	M	iii	NP	4
39.PKP	7.5	L	iii	G	2,3
40.RD	7.8	L	iii	G	-
41.SR	7.8	H	i/ii	np	6
42.NPS	7.10	H	iv	NP	1
43.HN	7.6	L	iv	-	1
44.PP	7.5	M	i	0	6
45.SL	7.1	H	ii	NP	6
46.RL	7.2	M	iv	G	1
47.AS	7.3	H	iv	NP	1

48.MS	7.1	H	iv	NP	1
49.AC	7.2	H	iv	NP	1
50.TAG	7.3	M	iii	NP	3
51.APL	7.0	M	iv	NP	1
52.SL	7.10	H	iv	SFL	1
53.SS	7.5	H	iv	G	1
54.SH	7.6	H	iv	SFL	1
55.RK	7.9	L	iv	-	1
56.NR	7.2	M	i	-	6
57.AA	7.9	H	iv	-	1
58.SR	7.9	L	iv	G	1
59.RP	7.8	L	iv	-	1
60.SR	7.8	L	iii	NP	3
61.VK	7.1	L	i	-	-
62.KL	7.2	L	i	-	-
63.AA	7.6	M	iv	0	1
64.RK	7.3	M	i	-	-
65.JR	7.0	M	ii	-	-
66.AK	7.0	L	i	-	-
67.R	7.7	H	iv	-	1
68.ML	7.5	H	ii	0	-
69.NNK	8.3	M	iii	G	1
70.AS	8.0	M	ii	0	-
71.NTR	8.1	M	ii	-	-
72.RP	8.6	H	iii	NP	3
73.RP	8.6	H	iii	NP	3
74.AK	8.0	H	iv	NP	1
75.IRL	8.5	L	i/ii	0	6
76.SD	8.3	M	i/ii	0	6

77.AS	8.6	H	iv	NP	1	
78.PD	8.4	M	ii	-	5	
79.SK	8.7	M	ii	G	1	
80.M	8.8	H	ii	0	6	
81.KN	8.7	H	iii	-	1	
82.NS	8.4	H	ii	-	6	
83.AS	8.0	H	iv	0	1	
84.RK	8.2	M	iv	G	1	
85.GC	8.0	L	iv	-	1	
86.SKL	8.2	M	iv	-	1	
87.JN	8.4	H	iv	SFL	1	
88.RP	8.0	H	iv	NP	1	*
89.AB	8.6	M	iv	NP	1	
90.ST	8.6	H	iii	0	-	
91.AC	9.2	L	ii	-	-	
92.RP	9.5	M	i/ii	-	6	
93.SSD	9.3	M	ii	NP	-	
94.LD	9.0	M	ii	0	3	
95.R	9.11	H	iv	-	1	*
96.SM	9.10	H	iii	0	3	*
97.N	9.9	H	iv	SFL	1	*
98.NC	9.0	M	iv	0	1	
99.SP	10.7	H	iv	-	1	*
100.F	10.2	H	iv	-	1	*
101.RK	10.10	H	iv	-	1	*
102.M	10.11	H	iv	-	1	*
103.JH	10.2	H	iv	NP	1	*
104.AD	11.0	H	iv	-	1	*
105.N	11.4	H	iv	-	1	*

Appendix 5. The set of problems for the study of performance on 3-addend problems with operators other than addition-only

16 - 10 - 5

19 - 7 - 6

15 - 9 - 4

5 + 2 - 5

12 - 6 + 4

5 - 3 + 5

7 * 2 * 4

3 * 5 * 5

2 * 10 * 6

12 / 6 / 2

32 / 4 / 2

5 * 3 / 3

10 / 2 * 2

Appendix 6. Subjects' performance on 3-addend problems with other operators (besides addition-only)

subject	age	used grouping on
SP	10.7	* only
F	10.2	* only
R	9.11	* only
AD	11.0	* only
N	11.4	*, -, / only problems, but not on problems with combinations of operators.

The other 6 subjects did not use grouping for any of the problems.

Appendix 7. A listing of the program

```
;;; -*- Mode:Common-Lisp;Package:(RD (LISP));Base:10 -*-
(in-package 'RD :use '(lisp))

(defvar *world*)

(defvar *nots*)

(defvar *abstractions*)

(defvar *known-facts*)

(defun sstart (instance)
 (setq *world*
     (append instance
           (subst 'old-number 'number (subst 'old-used 'used *world*))))
 (rule-interpreter (reverse (apply-prods instance)) instance))

(defun rule-interpreter (cset instance)
 (cond
  ((problem-solved) 'done)
  ((null cset) '(no rules apply - learning here))
  (t (perform-action (conflict-resolution cset) instance)
    (rule-interpreter (remove-repeat cset  (reverse (apply-prods instance)))
              instance))))

(defun problem-solved ()
 (or (equal (length (remove-if-not #'(lambda (i) (equal (car i) 'number))
                    *world*))
       (1+ (length (remove-if-not #'(lambda (i) (equal (car i) 'used))
                        *world*)))) (equal (caar *world*)
                                  'answer)))

(defun remove-repeat (lis1 lis2)
 (cond
  ((equal (car lis1) (car lis2)) (cdr lis2))
  (t lis2)))

(defun conflict-resolution (cset)
```

```
  (let (recent-value recent-value-clauses )
    (setq recent-value (apply #'max (apply 'append (mapcar #'caar cset))))
     (setq recent-value-clauses (remove-if-not #'(lambda (i) (member recent-
value
                                                    (caar i)))
                                cset))
    (cond ((equal (length recent-value-clauses) 1)
           (car recent-value-clauses))
          (t (setq recent-value (apply #'max (delete 'nil (mapcar 'cadr
                 (mapcar 'nsort (mapcar 'caar recent-value-clauses))))))
           (setq recent-value-clauses (remove-if-not #'(lambda (i)
                                (equal recent-value (cadr (nsort
                                          (caar i)))))
                                       recent-value-clauses))
           (car recent-value-clauses)) )))

(defun nsort (li)
  (sort li '>))

(defun perform-action (actions instance)
  (declare (ignore instance))
  (map nil #'(lambda (i)
           (print (cdr i)) (setq *world* (append (list (cdr i)) *world*)))
      actions))

(defun apply-prods (instance)
  (do ((plist *abstractions* (cdr plist))
      (result nil
           (append result
                (match-prod (car plist)
                        (new-match (get-test (car plist))) instance))))
      ((null plist) result)))

(defun new-member (element list)
  (cond
   ((null list) nil)
   ((atom (car list))
    (cond
     ((equal element (car list)) list)
     (t (new-member element (cdr list)))))
   (t
    (cond
```

```
    ((new-member element (car list)))
    (t (new-member element (cdr list)))))))

(defun match-prod (prod bindings instance)
  (cond
   ((null bindings) nil)
   (t
    (append (match-prod prod (cdr bindings) instance)
        (remove nil
            (list
              (multiple-actions (act-pattern prod) (car bindings)
                    instance)))))))

(defun get-test (prod)
  (car prod))

(defun act-pattern (prod)
  (caddr prod))

(defun multiple-actions (actions bindings instance)
  (cond
   ((null actions) nil)
   (t
    (append (multiple-actions (cdr actions) bindings instance)
        (build-clause (car actions) bindings )))))

(defun build-clause (clause bindings)
  (do ((old-clause clause (cdr old-clause))
      (new-clause nil
            (append new-clause
                (list
                  (or (associate (assoc (car old-clause)
                                        (delete-nos bindings)
                                        :test #'equal))
                      (car old-clause))))))
     ((null old-clause)
     (edit-old
      (cond
       ((equal (car new-clause) 'fn) (eval (cdr new-clause)))
       (t new-clause))
      (car bindings)))))
```

```
(defun edit-old (clause recency-value)
  (cond ( (some #'(lambda (i) (or (member clause i) (equal clause i)))
           *world* ) nil)
        (t (list (cons recency-value clause)))))

(defun associate (lis)
  (cadr lis))

(defun new-match (pattern)
  (new-match1 pattern '(nil)))

(defun new-match1 (parts answers)
  (cond
   ((null parts) (fdups answers nil))
   ((null answers) nil)
   (t
    (new-match1 (cdr parts)
           (rd-merge
           (cond
            ((equal (car (car parts)) 'not)
             (cons 'not (match1 (cadr (car parts)))))
            ( (equal (car (car parts)) 'fn) (test-function
                                  (cdr (car parts))
                                  answers))
            (t (match1 (car parts))))
           answers)))))

(defun test-function (test-fn bindings)
  (cond ((equal (car test-fn) 'equal) (fequals bindings nil))
        (t (print '(test fn. not defined)))))

(defun fequals (lis result)
  (cond
   ((null lis) result)
   (t
    (fequals (cdr lis)
          (cond
           ((eqtest (cdar lis) nil)  (cons (car lis) result))
           (t  result))))))

(defun eqtest (lis res)
  (cond
```

```
   ((null lis) nil)
   ((member (next-value lis) res) t)
   (t (eqtest (cdr lis) (cons (next-value lis) res)))))

(defun fdups (lis result)
  (cond
   ((null lis) result)
   (t
    (fdups (cdr lis)
         (cond
          ((duptest (car lis)) result)
          (t (cons (car lis) result)))))))

(defun duptest (lis)
  (cond
   ((null lis) nil)
   ((numberp (caar lis))
    (cond
     ( (and (member (caar lis) (cdar lis) :test #'equal)
          (equal (length (delete (caar lis) (car lis))) 0)) t)
    (t (duptest (cdr lis)))))
   (t (duptest (cdr lis)))))

(defun next-value (lis)
  (cadar lis))

(defun nos-first (lis)
  (mapcar #'check-number lis))

(defun check-number (lis)
  (append (list (copy-seq (remove-if-not #'numberp lis)))
        (delete-nos lis)))

(defun delete-nos (lis)
  (remove-if #'numberp lis))

(defun merge-old (ins previous)
  (apply #'append
        (mapcar #'(lambda (y) (merge-pair ins y)) previous)))

(defun merge-pair (ins1 ins2)
  (cond
```

```
   ((or (null ins1) (null ins2)) (list (or ins1 ins2)))
   (t
    (do ((chores ins1 (cdr chores))
         (result ins2 (merge-ins result (car chores))))
        ((null chores)
         (cond
          (result (list result))
          (t nil)))
      (cond
       ((null result) (return nil)))))))

(defun merge-ins (ins pair)
  (cond ((numberp (car pair)) (append (list (cons (car pair) (car ins)))
                                      (cdr ins)))
        ((and (equal (length ins) 1) (numberp (caar ins))) nil)
        ((numberp (caar ins)) (cond ((null (merge-ins (cdr ins) pair)) nil)
                                    (t (append (list (car ins)) (merge-ins
                                                                 (cdr ins) pair)))))
        ((assoc (car pair) ins :test #'equal)
         (cond
          ((equal (cadr pair) (cadr (assoc (car pair) ins :test #'equal)))
           ins)
          (t nil)))
        (t (cons pair ins))))

(defun elem-match (pattern data)
  (cond
   ((variablep pattern) (list pattern data))
   ((equal pattern data) t)
   (t nil)))

(defmacro nthchar (x n)
  `(intern (subseq (princ-to-string ,x) (1- ,n) ,n) 'rd))

(defun variablep (term)
  (equal (nthchar term 1) '=))

(defun attach-recency (lis dat &optional (world (reverse *world*)) (index 1))
  (when world
    (if (equal (car world) dat)
      (cons (cons index lis)
            (attach-recency lis dat (cdr world) (1+ index)))
      '
```

```
    (attach-recency lis dat (cdr world) (1+ index)))))

(defun delete-ts (lis)
  (remove 't lis))

(defun match1 (pattern &optional (world (reverse *world*)) (index 1))
  (when world
    (append (match-clause pattern (car world) index)
        (match1 pattern (cdr world) (1+ index)))))

(defun match-clause (pat dat position)
  (cond
   ((equal (length pat) (length dat))
    (filter (mapcar 'elem-match pat dat) position))
   ((and (listp (car dat)) (equal (length pat) (length (car dat))))
    (filter (mapcar 'elem-match pat (car dat)) position))
   (t nil)))

(defun filter (lis position)
  (cond
   ((member nil lis :test #'equal) nil)
   (t (list (cons (list position) (delete-ts lis))))))

(defun rd-merge (new previous)
  (cond
   ((equal (car new) 'not) (setq *nots* (append (cdr new) *nots*))
    (merge-nots *nots* previous))
   ((null new) nil)
   ((equal previous '(nil)) new)
   (t (apply #'append
         (mapcar #'(lambda (x) (merge-old  x previous)) new)))))

(defun merge-nots (*nots* previous)
  (let ((merged-list
      (car
       (last
        (remove 'nil
            (mapcar
             #'(lambda (j)
                (when (some #'(lambda (i) (each-not (cdr i) j)) *nots*)
                 (setq previous (remove j previous))))
             previous))))))
```

```
  (cond
   ((null merged-list) previous)
   (t merged-list))))

(defun each-not (not-item previous-item)
  (not (member nil (mapcar #'(lambda (i) (member i previous-item :test
#'equal))
                    not-item) :test #'equal)))

(defun coladd (i1 a i2 b)
 (cond
  ((> a b) (list 'addd i1 a i2 b))
  (t (list 'addd i2 b i1 a))))

(defun addd (addend1 addend2)
 (do ((total addend1 (+ total 1)) (counter 0 (+ counter 1)))
    ((= counter addend2) (list 'number 0 total))))

(defun caladd (i1 a i2 b)
 (cond
  ((> a b) (list 'addcal i1 a i2 b))
  (t (list 'addcal i2 b i1 a))))

(defun addcal (addend1 addend2)
 (do ((total 0 (+ total 1)) (counter 0 (+ counter 1)))
    ((= counter (+ addend1 addend2)) (list 'number 0 total))))

(defun cafadd (addend1 addend2)
 (setq fract (* .1 addend2))
 (do ((total 0 (+ total 1)) (counter 0 (+ counter 1)))
    ((= counter (+ addend1 addend2)) (list 'number 0 total (+ fract total)))))

(defun lookup-first (a b c)
 (cond
  ((lookup a b) (list 'lookup 'success (+ a b) c))
  (t (list 'lookup-first 'unsuccess a b c))))

(defun lookup-second (a b c)
 (cond
  ((lookup a c) (list 'lookup 'success (+ a c) b))
  (t (list 'lookup-second 'unsuccess a b c))))
```

```
(defun lookup-third (a b c)
  (cond
   ((lookup b c) (list 'lookup 'success (+ b c) a))
   (t (list 'lookup-third 'unsuccess a b c))))

(defun lookup (x y)
  (car (nth (1+ y) (car (nth (1+ x) *known-facts*)))))

(defun doit (a b c)
  (addd (addd a b) c))

(setf *known-facts*
    '((0 1 2 3 4 5 6 7 8 9) (1 2 3 4 5 6 7 8 9 10) (2 3 4 5 6 7 8 9 10 11)
      (3 4 nil 6 nil nil nil nil nil nil) (4 5 nil nil 8 nil nil nil nil nil)
      (5 6 nil nil nil 10 nil nil nil nil)
      (6 7 nil nil nil nil 12 nil nil nil)
      (7 8 nil nil nil nil nil 14 nil nil) (8 9 nil nil nil nil nil nil 16 nil)
      (9 10 nil nil nil nil nil nil nil 18)))

(defun match-all-conds (rules)
  (cond ((null rules) nil)
        (t (append (list (match-yes-or-no (get-test (car rules))))
                   (match-all-conds (cdr rules)) ))))

(defun  match-yes-or-no (conds)
  (cond (( null conds) nil)
        (t (append (match2 (car conds)) (match-yes-or-no (cdr conds)) ))))

(defun match2 (1cond)
  (cond ((equal (car 1cond) 'not) (cond ( (match1 (cadr 1cond)) '(no))
                                        (t '(yes))))
        (t (cond ((match1 1cond) '(yes))
                 (t '(no))))))

(defun remove-cond (*abstractions*)
  (setq *abstractions*  (cons (cons (delete (nth (position
                                                  'no (car (match-all-conds
                                                            *abstractions*)))
                                                 (caar *abstractions*))
                                            (caar *abstractions*) :test 'equal)
                                    (cdar *abstractions*)) (cdr *abstractions*))))
```

```
(defun check-operator (operator a b c)
  (cond ((equal operator '(equal x y)) (or (equal a b) (equal b c) (equal a c)))
        ((equal operator '(equal (+ x y) z)) (or (equal (+ a b) c) (equal (+ b c)
                                                                       a) (equal (+ a c) b)))
        ((equal operator '(equal (+ x y) 10)) (or (equal (+ a b) 10) (equal (+ b
                                                                            c) 10)
                                                  (equal (+ a c) 10)))
        ((equal operator '(equal x 1)) (or (equal a 1) (equal b 1) (equal c 1)))
        ((equal operator '(equal (+ 1 x) y)) (or (equal (+ 1 a) b) (equal
                                                                   (+ 1 b) c)
                                                 (equal (+ 1 a) c) (equal
                                                                   (+ 1 b) a)
                                                 (equal (+ 1 c) a) (equal
                                                                   (+ 1 c) b)))))

(defun learn-cond (operators a b c)
  (cond ((null operators) *abstractions*)
        ((check-operator (car operators) a b c) (setq *abstractions* (append
          (list (append (list (cons (car operators) (caar *abstractions*)))
           (cdar *abstractions*))) (cdr *abstractions*))))
        (t (learn-cond (cdr operators) a b c))))
```